MASTER EN SCIENCES INFORMATIQUES

Développement d'un outil de surveillance et de détection d'attaques sur réseaux Ethernet

Regaya, Borhène

Award date:
1994

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE DAME DE LA PAIX
NAMUR

Institut d'Informatique
Rue Grandgagnage, 21, 5000 Namur

DEVELOPPEMENT D'UN OUTIL DE SURVEILLANCE ET DE DETECTION D'ATTAQUES SUR RESEAUX ETHERNET

Borhène REGAYA

Promoteur: Monsieur Néji Habra

Mémoire présenté en vue de l'obtention du grade de
Licencié et Maître en informatique

Année académique 1993-1994

Remerciements

A cet endroit, je voudrais adresser un mot de remerciement à toutes les personnes qui ont contribué à la rédaction de ce mémoire.

Tout d'abord je voudrais remercier mon promoteur Monsieur Néji Habra, promoteur de ce mémoire, pour le suivi de ce travail. Ses commentaires et ses questions critiques m'ont été très utiles.

Je suis également reconnaissant envers Monsieur Aziz Mounji pour sa disponibilité, son aide et ses explications concernant la programmation, ses lectures et son soutien.

Je tiens à remercier ma famille pour toute leur attention à mon égard.

Enfin, je tiens à marquer ma gratitude à U. Souren ainsi que tous mes amis qui m'ont aidé à rédiger ce mémoire, ils m'ont fait part de remarques pertinentes sur presque toutes ses pages.

Résumé

Le projet ASAX (*Advanced Security Audit Trail Analysis on uniX*) a été développé conjointement par une équipe de l'Institut d'Informatique et la société Siemens. Il a pour objectif de définir un outil d'analyse se basant sur un fichier séquentiel de traces généré par le système d'exploitation, et sur un langage de requêtes RUSSEL (*RUle-baSed Sequence Evaluation Language).* Ce langage va se servir de ce fichier de traces, traduit en un format NADF (*Normalized security Audit Data Format*), comme base de données pour pouvoir répondre aux requêtes en faisant une analyse en un seul passage.
Les stations de travail, et en particulier les stations Sun Sparc, connectées sur le réseau local de l'Institut d'Informatique peuvent capter des paquets passant sur Ethernet et les garder éventuellement dans les fichiers de traces. Ces fichiers peuvent être traduits en un format approprié et, par la suite, analysés par le langage RUSSEL qui est basé sur des requêtes.
Les protocoles utilisés dans le réseau Ethernet sont ceux du modèle TCP/IP. Ce modèle a été conçu pour connecter différents types de support de communication utilisant la technique du paquet. Ces supports sont reliés entre eux par des passerelles. Le modèle TCP/IP fournit un service fiable de communication entre processus.
Dans le cadre du mémoire, un adaptateur de format a été implémenté dans le but de traduire les fichiers de traces générés par le système d'exploitation en un fichier de format NADF, en respectant tant les protocoles utilisés que le format approprié des fichiers de traces et les règles générales du langage RUSSEL. L'applicabilité de l'évaluateur ASAX à la détection d'utilisation malveillantes du système a été prouvée par l'élaboration de règles permettant de telles détections. Cet outil nous permettrait de surveiller et de détecter des attaques sur le réseau Ethernet.

Abstract

The ASAX (*Advanced Security Audit Trail Analysis on uniX*) project is jointly developped by a team of the Institute of computer science of Namur and Siemens. It aims at defining an analysis tool using a sequential audit trail file generated by the operating system and a rule based langage RUSSEL (*RUle-baSed Sequence Evaluation Language).* This langage will be applied on a NADF (*Normalized security Audit Data Format*), which is an adaptation of the audit trail already generated, as a database to answer the requests in one and only one pass.
The workstations, and in particular the Sun Sparc workstations, connected on the local area network of the Institute can capture packets through the Ethernet support and keep them possibly in some trail files. These files may be translated in an appropriate format, and then analysed by the rule based langage RUSSEL.
The protocols used on the Ethernet network are based on the TCP/IP model. This model was designed to interconnect different type of communication supports using the packet technologie. These devices are interconnected by means of gateways. The TCP/IP model provide a reliable communication service between processes.
During this thesis a format adaptor was implemented in order to translate the native trail files to NADF format, respecting the used protocols, the appropriate format of the trail files and the principal rules of the RUSSEL langage. The applicability of the ASAX evaluator to the detection of security breaches and misuses was then proven by implementing rules in RUSSEL for such detection. This tool will provide us with more control and detection of attacks on the Ethernet network.

Table des matières

INTRODUCTION 1

PARTIE I: ENVIRONNEMENT ET OUTILS 3

CHAPITRE I.1: ENVIRONNEMENT DE BASE 3
I.1.1. Ethernet 3
I.1.1.1. Vue générale 3
I.1.1.1.1. Objectif 3
I.1.1.1.2. Les différents composants 3
I.1.1.1.3. Les différentes topologies 4
I.1.1.1.4. Les protocoles de base 5
I.1.1.1.5. Les versions 5
I.1.1.2. Les composants d'Ethernet 6
I.1.1.2.1. Le support physique: les câbles 7
I.1.1.2.1.1. Les câbles standards 10BASE5 7
I.1.1.2.1.2. Les câbles "*thinnet*" 10BASE2 8
I.1.1.2.1.3. Les paires torsadées 10BASET 9
I.1.1.2.2. Les composants du support Ethernet: terminateurs, connecteurs, et adaptateurs ou raccords 9
I.1.1.2.3. Les émetteurs-récepteurs et les robinets 10
I.1.1.2.4. Le câble de raccordement (*drop cable*) 11
I.1.1.2.4.1. Les connecteurs 11
I.1.1.2.4.2. L'affectation des épingles (*pins*) 11
I.1.1.2.5. La connexion et l'expansion des réseaux Ethernet 12
I.1.1.2.5.1. Les répéteurs 13
I.1.1.2.5.2. Les ponts (*bridges*) 14
I.1.1.2.5.3. Les routeurs 15
I.1.1.2.5.4. Les passerelles 15
I.1.1.3. Les normes 15
I.1.1.4. Les échanges 16
I.1.1.5. Les limites et les tailles d'Ethernet 16
I.1.1.6. Conclusion 17

I.1.2. Unix 18
I.1.2.1. Présentation d'UNIX 18
I.1.2.1.1. Historique 18
I.1.2.1.2. Versions récentes d'UNIX 18
I.1.2.1.3. Fonctionnalités 19
I.1.2.2. La structure du système UNIX 20
I.1.2.2.1. Le noyau 21
I.1.2.2.2. Le shell 22
I.1.2.2.3. Les outils UNIX 23
I.1.2.3. Les utilisateurs 23
I.1.2.3.1. Accès au système 23

I.1.2.3.2. Sortie du système 24
I.1.2.3.3. Contexte 24
I.1.2.4. Organisation des fichiers UNIX 25
I.1.2.4.1. Arborescence d'UNIX 25
I.1.2.4.2. Attribut des fichiers 26
I.1.2.4.3. Organisation traditionnelle 26
I.1.2.5. Les processus 28
I.1.2.5.1. Notion de processus 28
I.1.2.5.2. Mécanismes 29
I.1.2.5.3. Communication entre processus 29
I.1.2.6. Caractéristiques d'UNIX 31
I.1.2.6.1. Capacité multitâches 32
I.1.2.6.2. Capacité multi-utilisateurs 33
I.1.2.6.3. Portabilité du système UNIX 33
I.1.2.6.4. Les programmes offerts par UNIX 33
I.1.2.6.5. Les communications sur UNIX 34
I.1.2.6.6. Les programmes d'applications de tiers 35
I.1.2.7. SunOS 35
I.1.2.7.1. Introduction 35
I.1.2.7.2. Les services réseau 35
I.1.2.7.3. Terminologie 36
I.1.2.7.4. Les services réseaux dominants 36

CHAPITRE I.2: LA SECURITE ET SES OUTILS 41
I.2.1. Introduction: Le problème de la sécurité en général 41
I.2.1.1. La nécessité de la protection 41
I.2.1.2. La qualité souhaitée 41
I.2.1.3. Les différents types de dangers 41
I.2.2. Le problème de la détérioration des données 43
I.2.2.1. Le problème 43
I.2.2.2. Quelques approches de protection 43
I.2.2.3. Une classification des niveaux de protection 43
I.2.3. Le problème de l'alimentation 46
I.2.4. Les attaques malicieuses 48
I.2.4.1. Les différents types d'attaques 48
I.2.4.2. Les différents remèdes 49
I.2.4.3. Les différents critères d'évaluation 50
I.2.5. La génération et l'analyse des traces 52
I.2.5.1. L'audit de sécurité 52
I.2.5.2. Les événements et la sélection 53
I.2.5.3. Les fichiers de traces 53
I.2.6. Asax: Un outil d'analyse de traces 55
I.2.6.1. Introduction 55
I.2.6.2. Les objectifs 57
I.2.6.3. Les problèmes principaux 57
I.2.6.3.1. La variété de scénarios dans la sécurité 57
I.2.6.3.2. La réutilisabilité, la généricité et l'universalité 58
I.2.6.3.3. L'interface utilisateur 58
I.2.6.4. Les qualités d'Asax 58

I.2.6.4.1. L'universalité 58
I.2.6.4.2. La puissance: le langage RUSSEL 59
I.2.6.4.3. L'efficacité: l'implémentation 61
I.2.6.4.4. La portabilité 63
I.2.6.4.5. Autres caractéristiques d'Asax 63

PARTIE II: LE MODELE TCP/IP 64

CHAPITRE II.1: DEFINITION ET ARCHITECTURE 64
II.1.1. Introduction 64
II.1.1.1. Définition générale 64
II.1.1.2. Les objectifs 64
II.1.1.3. Historique 65
II.1.2. Principes et fonctionnement de TCP/IP 65
II.1.2.1. Les protocoles: objectifs et définitions 65
II.1.2.2. Architecture d'une famille de protocoles: les couches connceptuelles 66
II.1.2.3. L'architecture en couches du modèle TCP/IP 68
II.1.3. TCP/IP face à OSI 72
II.1.3.1. L'architecture en couches du modèle OSI 72
II.1.3.2. Comparaison des deux modèles 75
II.1.3.3. Les principales différences entre les deux modèles 77
II.1.3.3.1. Bout à bout contre fiabilité niveau liaison 77
II.1.3.3.2. Le contrôle 78
II.1.3.4. Migration vers le standard OSI 78

CHAPITRE II.2: L'ADRESSAGE INTERNET ET LES PROTOCOLES ARP ET RARP 80
II.2.1. L'adressage Internet 80
II.2.1.1. Introduction 80
II.2.1.2. Les adresses Internet 80
II.2.1.3. Faiblesse de l'adressage Internet 82
II.2.2. La résolution du problème des adresses (ARP) 83
II.2.2.1. Objectifs 83
II.2.2.2. La structure statique du protocole ARP 83
II.2.2.2.1. L'encapsulation du message ARP 83
II.2.2.2.2. Le format du protocole ARP 84
II.2.2.3. La résolution par liaison dynamique 85
II.2.2.4. La sauvegarde des correspondances des adresses 85
II.2.2.5. La relation entre ARP et les autres protocoles 86
II.2.2.6. L'implémentation de ARP 86
II.2.3. La détermination de l'adresse Internet au démarrage (RARP) 87
II.2.3.1. Introduction 87
II.2.3.2. Reverse Address Resolution Protocol (RARP) 87
II.2.3.3. Les problèmes de transactions avec RARP 89
II.2.3.4. Les serveurs RARP 89

CHAPITRE II.3: DESCRIPTION ET FONCTIONNEMENT DU SERVICE IP 91
II.3.1. Objectifs et problèmes 91
II.3.2. La structure statique: l'unité de transfert 92
II.3.2.1. Le datagramme 92
II.3.2.2. Formats et contenus des champs 92
II.3.3. Le fonctionnement dynamique 99
II.3.3.1. La fragmentation 99
II.3.3.2. Le routage des datagrammes 100
II.3.3.3. Les types de routage 100
II.3.3.3.1. Le routage directe 101
II.3.3.3.2. Le routage indirecte 101
II.3.3.3.3. Les algorithmes de routage 102
a/ Le routage avec des tables 102
b/ Les routes par défaut 103
c/ Les routes spécifiques aux hôtes 103
d/ L'algorithme générale 104
II.3.3.3.4. La manipulation des datagrammes à l'arrivée 104

CHAPITRE II.4: LE PROTOCOLE ICMP 106
II.6.1. Objectifs et problèmes 106
II.6.1.1. Les objectifs 106
II.6.1.2. Les problèmes 106
II.6.2. Structure statique 107
II.6.2.1. Le système de livraison 107
II.6.2.2. le format du message ICMP 107
II.6.3. structure dynamique 108
II.6.3.1. Les tests d'atteinte d'une destination 108
II.6.3.2. Le contrôle de flux 109
II.6.3.3. Les changements de routes 110
II.6.3.4. La résolution du problème de routage 111
II.6.3.5. La résolution du problème des paramètres d'en-tête 112
II.6.3.6. La synchronisation de l'horloge et l'estimation du temps de transit 112
II.6.3.7. L'obtention d'une adresse réseau 113

CHAPITRE II.5: LE PROTOCOLE UDP 114
II.6.1. Objectifs 114
II.6.2. Structure statique 114
II.6.2.1. Les ports 114
II.6.2.2. La structure du protocole 115
II.6.2.3. Le format UDP 115
II.6.3. Aspect dynamique 117
II.6.3.1. UDP et les couches de protocole 117
II.6.3.2. Le calcule de la somme de contrôle et le modèle en couche 117
II.6.3.3. Le démultiplexage UDP 117

CHAPITRE II.6: DESCRIPTION ET FONCTIONNEMENT DU SERVICE TCP 119
II.6.1. Objectifs et problèmes 119
II.6.1.1. Les objectifs 119
II.6.1.2. Les problèmes de livraison 120
II.6.1.3. Liaison entre l'interface d'application et service fiable de livraison 120
II.6.2. L'aspect statique de la structure 122
II.6.2.1. Segments, séquences, et numéros de séquences 122
II.6.2.2. La variation de la taille de la fenêtre et le contrôle de flux 122
II.6.2.3. Format du segment TCP 123
II.6.3. L'aspect dynamique 126
II.6.3.1. La fiabilité 126
II.6.3.2. La technique de fenêtrage 127
II.6.3.3. Le calcul de la somme de contrôle 129
II.6.3.4. Accusé de réception et retransmission 129
II.6.3.5. Le délai d'attente et la retransmission 130
II.6.3.6. La réponse à l'encombrement 130
II.6.3.7. L'établissement d'une connexion 131
II.6.3.8. La fermeture d'une connexion 132
II.6.3.9. Remise à zéro de la connexion 133
II.6.3.10. Livraison forcée 133
II.6.3.11. Les numéros de "port" réservés 134

CHAPITRE II.7: LE FONCTIONNEMENT DU COUPLE TCP/IP 135
II.7.1. Objectifs et problèmes 135
II.7.1.1. Introduction 135
II.7.2. Fonctionnement 136
II.7.2.1. Etablissement d'une connexion 136
II.7.2.2. Transfert de données 139

PARTIE III: IMPLEMENTATION 141

CHAPITRE III.1: LE FICHIER NADF 141
III.1.1. L'adaptateur de format 141
III.1.1.1. Introduction 141
III.1.1.1.1. Objectifs 141
III.1.1.1.2. Généralités sur les adaptateurs de format 141
III.1.1.2. Structure du fichier NADF: Format des enregistrements 141
III.1.1.3. Le fichier auxiliaire du fichier NADF: description des audits data 143
III.1.1.4. Description et implémentation de l'adaptateur de format 145
III.1.2. Les problèmes rencontrés 147
III.1.2.1. Au niveau de la programmation 147
III.1.2.2. Au niveau de l'analyse *on-line* 148

CHAPITRE III.2: LES REQUETES RUSSEL 150
III.2.1. L'analyse des fichiers de traces 150
III.2.1.1. Introduction 150
III.2.1.2. L'interface avec le langage C 151
III.2.1.3. Les règles RUSSEL 152

CONCLUSION 154

GLOSSAIRE 155

BIBLIOGRAPHIE 162

ANNEXE A: EXEMPLES DE REGLES RUSSEL 165
ANNEXE B: LA SYNTAXE ET LA SEMANTIQUE DU LANGAGE RUSSEL 169
ANNEXE C: DESCRIPTION DE LA COMMANDE ETHERFIND 178
ANNEXE D: EXEMPLE DE FICHIER DE TRACES 181
ANNEXE E: PROGRAMME DE L'ADAPTATEUR DE FORMAT 183
ANNEXE F: PROGRAMMES DES REQUETES RUSSEL 184

Introduction

Pour atteindre un niveau de sécurité idéal, un système informatique doit prouver que son système d'exploitation, son logiciel de réseau et son matériel sont bien protégés contre tout genre d'attaques (intrusion, virus etc.). Malheureusement, les systèmes actuels sont loins de ressembler à cet idéal.

Certains systèmes d'exploitation peuvent générer une description des aspects sélectionnés des informations qui passent sur le réseau dans des fichiers de traces. Ainsi, l'analyse intelligente de ces fichiers constitue une tâche importante de la gestion de la sécurité du système.

Un fichier de traces est ainsi un ensemble de description présélectionné des événements de sécurité sous forme de paquets écrit par le mécanisme d'audit dans un ou plusieurs fichiers. Après sa création, le fichier de traces peut être analysé, l'auditeur autorisé peut extraire les informations de sécurité sélectionnées.

Certaines machines Unix connectées sur le réseau de l'Institut d'Informatique sont capables de détecter tous les paquets passant sur le réseau et de les garder éventuellement dans un fichier. Pour cela, il faut exécuter la commande "etherfind" avec les options appropriées. Cette commande a pour objectif de capter des paquets sur le réseau Ethernet. Elle est disponible dans le logiciel du réseau comme plusieurs autres options, mais pour des raisons de sécurité, il faut disposer de certains privilèges ou être super utilisateur pour pouvoir l'exécuter.

Un logiciel a été développé à l'institut en collaboration avec la société Siemens dans le but d'analyser tous les fichiers de traces en une et une seule passe. Ce logiciel est l'outil ASAX. ASAX permet l'analyse des différents fichiers de traces générés par le système d'exploitation en fournissant à l'utilisateur un langage d'interrogation simple permettant d'exprimer des requêtes d'analyse sur le fichier concerné. Le fichier à analyser devrait être d'abord mis dans un format standard simple reconnu par ASAX. Ce format est appelé le format NADF (pour *Normalized security Audit Data Format*).

Ce mémoire a pour objectif de mettre au point un système de sécurité qui contrôle et protège les systèmes informatique et en particulier les informations qui circulent sur le réseau Ethernet de l'institut. La première étape consiste à implémenter un adaptateur de format qui assure la conversion du fichier généré par le système d'exploitation (la commande etherfind) en un format NADF; ensuite, appliquer ASAX pour l'analyse de ce fichier. Des requêtes RUSSEL pertinentes devrait être également implémentées pour détecter les événements malicieux. Mais avant de présenter ces programmes, il nous a semblé intéressant de situer le contexte général dans lequel se développent ces différents outils et concepts.

C'est ainsi que dans la première partie nous présenteront une brève description de l'environnement et des outils utilisés. L'environnement de base est constitué du système d'exploitation Unix et en particulier SunOS, et du support Ethernet. Ces deux environnements sont à la base de l'analyse des fichiers de traces sur lesquels nous allons travailler. Ils sont

présentés dans le premier chapitre de cette partie. La sécurité et ses outils sont présentés dans le deuxième chapitre de cette partie (cf. I.2.). Nous présenterons dans ce chapitre les problèmes de sécurité en général (physiques et/ou logiques), ainsi que quelques types d'attaques, puis nous ferons une description non exhaustive de l'outil ASAX et du langage RUSSEL.

La deuxième partie sera consacrée à la présentation des concepts de protocoles et en particulier des protocoles du modèle TCP/IP qui constituent le réseau de l'Institut. Nous parleront également des mécanismes de connexion et de la différences qui existe entre ce modèle et le modèle OSI.

Dans le premier chapitre de cette partie nous présenterons une définition générale du modèle TCP/IP ainsi qu'une comparaison avec le modèle OSI. Le deuxième chapitre traitera de l'adressage Internet ainsi que des protocoles ARP et RARP. Le troisième chapitre, une description assez approfondie du service IP et de son fonctionnement; ce protocole étant à la base de l'analyse des paquets. Dans le quatrième chapitre, le protocole ICMP sera décrit. Le chapitre cinq présentera le protocole de transport UDP qui est également présent dans l'analyse des paquets. Le protocole TCP est présenté dans le chapitre six, c'est le coeur du modèle TCP/IP. Enfin, le chapitre sept traitera du fonctionnement du couple TCP/IP.

Cette partie utilise parfois des termes techniques, nous avons essayé pour notre part d'utiliser autant que possible des traductions françaises quand nous les connaissions. Toutefois, quand le mot anglais est utilisé le plus souvent, ou quand la traduction n'est pas aussi explicite, nous donnons entre parenthèses le mot anglais correspondant. Tous les mots en anglais seront écrits en italique. Il nous faut également noter que nous avons évité le plus possible d'utiliser des néologismes pour exprimer des termes spécifiques aux télécommunications. Néanmoins, quelques uns forts usités se sont certainement glissés dans le texte.

La troisième partie de ce mémoire est consacrée à l'implémentation de l'adaptateur de format (chapitre 1) et des requêtes RUSSEL établies pour les filtrages et les détections des événements anormaux.

Enfin, une conclusion ainsi qu'un glossaire sont présentés suivie par des annexes. L'annexe A reprend quelques exemples de requêtes RUSSEL présentés dans la littérature. L'annexe B fournit une brève description de la syntaxe et de la sémantique du langage RUSSEL. L'annexe C présentera une description complète de la commande etherfind. L'annexe D présentera un exemple de fichier de traces généré par etherfind. Et enfin, l'annexe E contiendra le code source de l'adaptateur de format et l'annexe F celui des reqêtes RUESSEL.

Partie I: Environnement et outils

Chapitre I.1.

ENVIRONNEMENT DE BASE

I.1.1 ETHERNET

I.1.1.1. Vue générale

I.1.1.1.1. Objectifs

Un réseau local est composé d'ordinateurs personnels, de postes de travail, de périphériques et de logiciels, le tout relié par des câbles, des cartes de communication et des programmes de gestion de réseau. Ethernet comprend uniquement les logiciels et les supports de communication; il permet la livraison de données grâce à la technique de diffusion et utilise les protocoles CSMA/CD. Il est composé d'un câble coaxial passif qui permet d'interconnecter tous les composants actifs.

Ethernet est un produit d'expérimentation qui a été conçu à l'origine par XEROX, DEC (*Digital Equipment Corporation*) et Intel. Il a été développé pour une technologie de réseau ayant une vitesse de 3 Mbits/sec; toutefois de nos jours la majorité des installations Ethernet utilisent la technologie la plus récente qui offre des vitesses de 10 Mbps (Méga bits par seconde).

Nous allons seulement exposer la technologie la plus récente. Tous ces éléments seront developpés de façon plus approfondie dans les sections qui suivent.

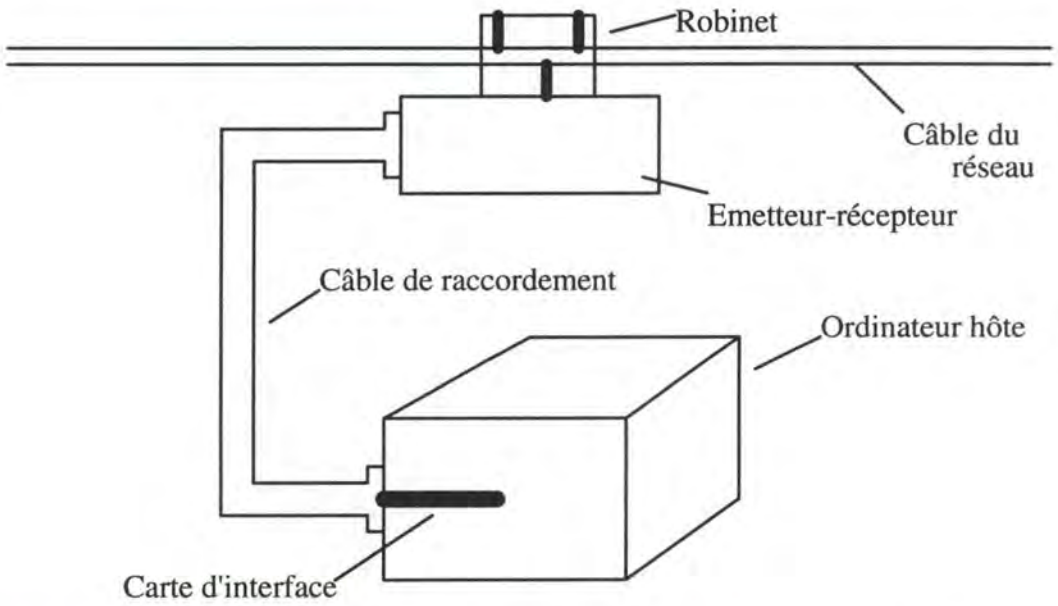
I.1.1.1.2. Les différents composants

Une installation Ethernet est constituée du câble de réseau, dont chaque extrémité est finie, et comporte des connexions avec chaque hôte du réseau. Chaque connexion est composée d'un robinet, d'un émetteur-récepteur ou unité d'accès média (MAU pour "*Media Access Unit*"), d'une carte d'interface dans l'ordinateur hôte, et d'un câble de raccordement qui relie l'émetteur-récepteur à la carte d'interface.

En fonction de la topologie, du câblage, des fonctions du poste (serveur ou station de travail), du coût,... nous déterminons la carte de communication adéquate.

La figure ci-dessous illustre une connexion Ethernet typique:

Diagramme de l'interface, l'émetteur-récepteur et du robinet Ethernet:



Les câbles utilisés dans une topologie en bus sont des câbles coaxiaux. Ces câbles coaxiaux sont constitués de:

- un câble conducteur central faisant office de conducteur principal,
- une armature métallique tressée constituant le conducteur de masse (retour du signal) et réalisant également un blindage à l'encontre des ondes électromagnétiques.

Dans ce type de réseau le câble coaxial est le plus employé. Cette topologie est la plus répandue car elle présente bon nombre d'avantages. Des détails plus approfondis seront donnés dans les sections suivantes.

I.1.1.1.3. Les différentes topologies

Il existe plusieurs topologies: le bus, l'étoile, l'anneau etc. Le choix de l'une ou de l'autre sera determiné par la vitesse à laquelle on veut travailler, par la disposition des lieux, par le type de câble que l'on désire utiliser ou qui est déjà installé, par les applications, etc.

En fait, tous ces facteurs sont interdépendants car dans certains cas plusieurs types de topologies peuvent être employés simultanément. La topologie de type bus est probablement la plus répandue à l'heure actuelle. Il s'agit d'une structure simple où chaque machine est reliée à un seul et même câble. Nous appellerons "noeud" chaque connexion avec le câble.

I.1.1.1.4. Les protocoles de base

Ethernet supporte donc une topologie en bus et utilise au niveau matériel le protocole à contention avec détection de collisions dit CSMA-CD (*Carrier Sense, Multiple Access with Collision Detection*).

Une topologie de branchement en bus ne permet pas de boucles; c'est-à-dire, qu'il n'y a qu'un seul chemin pour la transmission des paquets entre deux hôtes sur un même réseau.

Le protocole CSMA-CD est analogue à une réunion autour d'une table ronde où chacun des interlocuteurs (ordinateurs) peut prendre la parole; pour la prendre il attend d'abord une accalmie dans la conversation (aucun trafic sur le câble du réseau). Si deux interlocuteurs commencent à parler en même temps (*collision*), ils arrêtent la communication, écoutent sur le réseau, attendent un peu, puis l'un d'entre eux recommence à parler.

Le délai réel d'attente pendant une détection de collision (*Collision Detection*) est presque aléatoire, évitant ainsi le scénario suivant:

- transmission simultané sur le réseau,
- détection de la collision,
- attente du même intervalle de temps,
- et retransmission à nouveau synchrone, c-à-d collision.

L'algorithme employé pour le calcul des délais est appelé "*Truncated binary exponential backoff*". Chaque fois qu'un hôte essaye de transmettre et détecte une collision, l'algorithme change un bit.

Le temps d'attente après une collision pour un énième essai est calculé comme suit:

- si (0 <= n <= 10), le délai est de 51.2 µsec multiplié par un nombre aléatoire, uniformément distribué, entre 0 et $(2^n - 1)$.
- si (11 <= n <= 15), l'intervalle reste de 0 à 1023.
- si (n > 15), la transmission est abandonnée.

L'unité de temps du délai utilisée, 51.2 µsec, est le temps nécessaire pour transmettre 512 bits.

I.1.1.1.5. Les versions

Ethernet est disponible en trois versions:

- version 1 (la spécification d'origine de DEC-Intel-Xerox 1980),
- version 2 (la spécification révisée de DEC-Intel-Xerox 1982),
- et l'IEEE 802.3 (le standard IEEE officiel, 1983).

Les trois versions peuvent fonctionner ensemble. Néanmoins pour un ordinateur hôte individuel relié à un réseau, son émetteur-récepteur, son câble de raccordement, et sa carte d'interface doivent tous être issus de la même version.

La majorité des nouveaux supports Ethernet utilisent le standard IEEE; les discordances de versions proviennent essentiellement des anciennes installations.

I.1.1.2. Les composants d'Ethernet

Un des composants du réseau dont on ne parle pas assez, est le câble, support de l'information circulant sur le réseau. En effet, l'accent est souvent mis sur la vitesse du processeur ou sur le type de carte de communication à employer, mais un câble non adapté à l'application ou à la disposition des locaux peut poser beaucoup plus de problèmes.

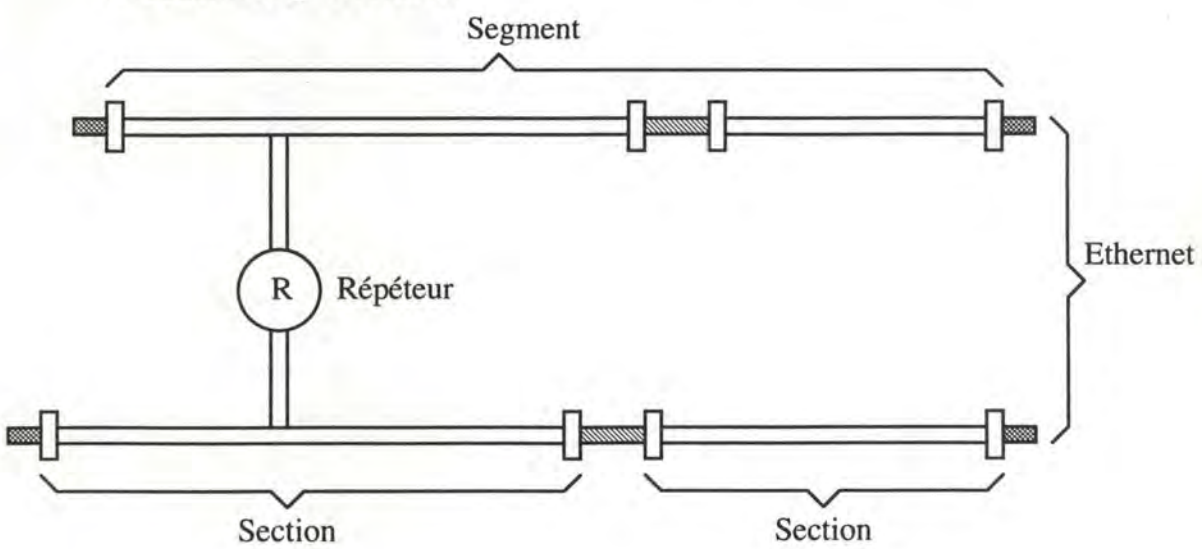
Le câble possède différents aspects dont il faut tenir compte. Ces aspects sont: la vitesse de transmission nécessaire ou désirée, les interférences éventuelles du monde extérieur et la sécurité des données circulant sur le média. Dans la majorité des cas, la disposition des lieux et aussi la topologie du réseau sont les facteurs décisifs pour le choix du média. Néanmoins tous ces facteurs sont plutôt interdépendants.

Dans ce qui suit nous décrivons les composants du matériel qui constituent l'installation Ethernet, et les limites que les standards Ethernet imposent à ces composants. On va utiliser la terminologie suivante pour illustrer les composants du câble Ethernet:

- une section de câble: la pièce physique continue d'un câble;
- un segment de câble: la pièce logique du câble (peut être constituée de plusieurs sections);
- un réseau logique: le réseau Ethernet constitué d'un groupe de segments reliés par un autre matériel.

Ces différents composants sont schématisés dans la figure ci-dessous:

La terminologie du câble:



I.1.1.2.1. Le support physique: les câbles

Les câbles utilisés actuellement sont de trois types: la paire torsadée, le câble coaxial et la fibre optique. Le plus ancien est la paire torsadée, aussi appelée "paire téléphonique" bien que le fil employé en informatique soit généralement mieux isolé. Cependant, le support physique standard d'Ethernet est le câble coaxial (10BASE5) , d'impédance 50Ω, et ayant une épaisseur d'un demi pouce (= 2.54 cm/2 = 1.27 cm) et des connecteurs de type N. Ce câble reste relativement cher.

Des spécifications et des interfaces existent maintenant pour des câbles coaxiaux (10BASE2) d'un quart de pouce (0.64 cm) avec des connecteurs BNC, très semblables à ceux utilisés pour les antennes de télévision, et également pour les terminaux à paire torsadé (10BASET) ou les fils téléphoniques; ces supports sont moins chers, mais leurs réseaux sont restreints à une fraction de la longueur totale spécifiée dans les standards Ethernet.

Les câbles Ethernet à bande large (10BROAD36), les câbles à fibre optique, et les canaux de transmission par Satellite peuvent être utilisés comme support physique, ainsi on arrive à des extensions considérables de la taille physique d'un Ethernet. Seule, la transmission par satellite n'a pas pu maintenir la vitesse Ethernet de 10 Mbits/sec, elle est limité à 56 Kbits/sec ou 1.5 Mbits/sec.

L'apparition des standards FDDI (*Fiber Distributed Data Interface*) avec une largeur de bande de 100 Mbits/sec et son MAN (*Metropolitan Area Network*) à grande vitesse, fournis par les compagnies téléphoniques promettent de dépasser les LAN (*Local Area Network*) dans les prochaines années.

Le câble coaxial, aussi appelé "coax" ou Ethernet, existe en plusieurs versions. Il faut tout d'abord faire la différence entre le "coax fin" (*thin Ethernet* ou *thinnet*) et le "coax gros" (*thick Ethernet* ou *thicknet*).

Le premier est le plus courant. Il se compose d'un conducteur central en cuivre, isolé par du plastique contenant lui même de l'air et d'un second conducteur qui peut être en cuivre ou en aluminium. Ce second conducteur est généralement sous la forme de torsade, il est concentrique par rapport au premier. L'ensemble est enrobé d'un isolant et d'une protection mécanique.

Le "*thicknet*" est un gros câble coaxial moins utilisé. Il se compose en réalité de plusieurs "coax fins", le tout entouré d'un isolant généralement de couleur jaune. Ce dernier est déjà employé dans des applications telles que le réseau de télévision, où il est nécessaire de véhiculer plusieurs types d'informations simultanément et sur des distances assez grandes.

I.1.1.2.1.1. Les câbles du standard 10BASE5

Le câble Ethernet standard peut être de deux types différentes, ayant soit une couche extérieure en PVC (*polyvinyl chloride* ou Polychlorure de vinyle) soit en Téflon. Si l'installation est en plein air, les recommandations de sécurité exigent que le câble PVC soit mis dans un conduit de protection. En effet, le PVC dégage des gaz toxiques quand il brûle.

Ces câbles PVC sont couramment appelé "tuyau jaune" à cause de leur couleur jaune brillante, le câble Téflon est souvent de couleur orange et est translucide. Le câble Téflon est un peu plus dure que le câble PVC, il a un diamètre d'un peu moins d'un demi pouce.

Les segments Ethernet doivent être constitués de sections de câble de longueurs standards. Les longueurs standards sont: 23.4 mètres, 70.2 mètres, et 117 mètres. Ces valeurs produisent des propriétés de transmission optimales. Il est également recommandé qu'ils soient mélangé de façon aléatoire. Nous remarquons que 70.2 et 117 sont des multiples de 23.4; ainsi , tous les câbles construits de sections de longueur standard doivent avoir une longueur totale divisible par 23.4 mètres.

Malheureusement, ceci implique parfois un usage inefficace du câble: par exemple, si on a besoin d'un segment de câble de 100 mètres, en utilisant les longueurs standards, on aura 17 mètres de câble qui resteront inutilisés.

Les segments Ethernet ont une longueur qui peut varier de 10 à 500 mètres. Ceci est contradictoire par rapport à ce qui précède, puisque ni 10 ni 500 mètres sont divisibles par 23.4.

La plupart des câbles Ethernet ont des marques noirs tous les 2.5 mètres. Ces marques indiquent les endroits sur lesquelles on peut brancher des émetteurs-récepteurs ou des connecteurs. Et comme la moitié de la longueur d'onde du signal est de 2.5 mètres, on place les dispositifs à ces points pour les rendre optimaux.

I.1.1.2.1.2. Les câbles "*thinnet*" 10BASE2

Les câbles "thinnet" ou "réseau bon marché" (*cheapernet*) fournissent également les deux types de supports: PVC et Téflon. Un réseau avec des segments "thinnet" a deux désavantages: la taille maximale du réseau est réduite, et chaque paire émetteur-récepteur/hôte devient une partie active du réseau et en cas de panne tout le réseau est mis hors d'usage. Ceci est analogue à tout circuit d'ampoules électriques que l'on met en série, si une ampoule est grillée tout le circuit tombe en panne.

Thinnet est moins cher et plus facile à manipuler. Il est surtout préféré dans les petites installations. "*Thinnet*" et "*thicknet*" peuvent être mêlés en utilisant des répéteurs ou des connecteurs. "*Thinnet*" est d'habitude de couleur noire, il ne contient donc pas des marques noires perceptibles; les émetteur-récepteur et les connecteurs sont ainsi montés de façon aléatoire.

La longueur maximale d'un câble contenant un segment "*thinnet*" est de 185 mètres. Les connecteurs en T qui placent l'émetteur-récepteur directement en série sur le câble du réseau sont utilisés à la place des câbles de raccordement. Les émetteurs-récepteurs doivent au moins se trouver à .5 mètre l'un de l'autre. Au plus 30 dispositifs peuvent être montés par segment de câble.

I.1.1.2.1.3. Les paires torsadées 10BASET

Le câble Ethernet à paire torsadée est relativement nouveau, et ses spécifications 10BASET ne sont pas encore devenues des standards IEEE.

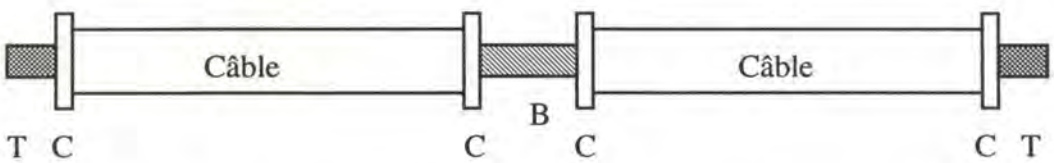
Un de ses grands attraits est la possibilité d'installer un Ethernet sur les lignes téléphoniques existantes, évitant ainsi le coût d'installation d'un nouveau câble. Les fils téléphonique à paires torsadées, comme le câble coaxial thinnet, sont moins chers et faciles à manipuler que les câbles Ethernet standards.

I.1.1.2.2. Les composants du support Ethernet: terminateurs, connecteurs, et adaptateurs ou raccords

Trois parties fondamentales sont nécessaires pour la construction des segments du câble Ethernet: le câble lui même, les connecteurs pour accoupler les extrémités des sections du câble, les adaptateurs à tambour pour joindre ensemble deux sections de câble, et les terminateurs pour aménager les extrémités.

Dans la figure ci-dessous deux sections d'Ethernet sont raccordées; le T indique la terminateur, C le connecteur, et B l'adaptateur en tambour. Les connecteurs sont attachés directement à l'extrémité du câble pour former une fiche (mâle) pour être branchée sur les adaptateurs ou les terminateurs. Les adaptateurs à tambour (*socket-socket*) se branchent à ces connecteurs pour joindre deux connecteurs de câble. Chaque segment logique Ethernet doit avoir à sa terminaison un inducteur d'impédance 50Ω appelé terminateur. Le terminateur (*terminator*) à pour but d'absorber tous les signaux qui se propagent dans le câble, ne permettant pas de réflexions. Les prises des connecteurs et les adaptateurs "*socket-socket*" existent mais ne sont pas standardisés.

Le segment du câble Ethernet:



On doit utiliser les fiches connectrices avec des prises adaptatrices à tambour et des terminateurs pour que les extrémités de chaque câble soient mâle ou femelle. Ceci facilite le dépannage quand il est nécessaire de supprimer logiquement (et pas physiquement) une section de câble. Bien que le dépannage est un peu plus facile, d'autres composants sont nécessaires pour chaque connexion utilisant ce système.

Par exemple, la figure ci-dessus montre une prise à tambour (B), des terminateurs (T), et des connecteurs de type fiche (C). Pour ce câble il a fallu utiliser deux sections de câble du réseau, deux terminateurs du type prise, un adaptateur à tambour "*socket-socket*", et quatre fiches connectrices. Chaque section du câble se termine par une fiche connectrice (C) et peut accepter ou bien une prise de type terminateur (T) ou bien un adaptateur à tambour "*socket-socket*" (B).

L'autre possibilité (brancher directement la fiche à la prise connectrice, ce qui économise ainsi un adaptateur) nécessite d'avoir à notre disposition des terminateurs à la fois mâles et femelles. Ce qui contredit les principes des spécifications.

I.1.1.2.3. Les émetteurs-récepteurs et les robinets

Un robinet est un dispositif qui se branche sur le câble du réseau, il est relié via le câble de raccordement à la carte d'interface de l'ordinateur hôte. Les émetteurs-récepteurs ne nécessitent pas une alimentation externe, ils s'alimentent plutôt par l'intermédiaire de l'interface de la machine hôte via le câble de raccordement. Il y a des petites différences entres les émetteurs-récepteurs suivant les versions.

La version 1 utilise des couplages DC avec un gaspillage élevé. La version 2 utilise un couplage AC sans gaspillage et un signal répétitif. L'IEEE 802.3 ajoute un nouveau type de contrôle appelé "*jabber control*" à la spécification de la version 2. Ce contrôle empêche le matériel qui tombe en panne d'inonder le réseau de faux paquets.

La version d'une carte d'interface peut être déterminée en vérifiant le voltage DC des deux tiges métalliques n° 3 et n° 10:

- Version 1- approximativement .7 volts;
- Version 2- approximativement .2 volts;
- IEEE 802.3- proche de 0 volts.

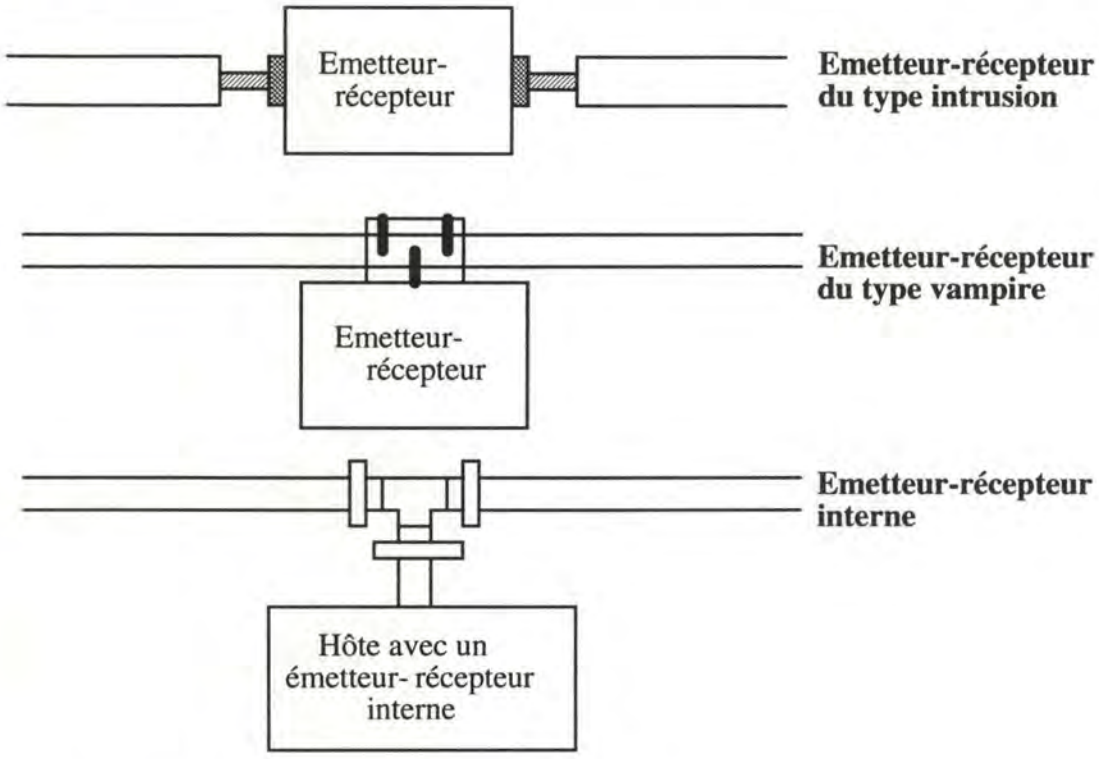
Il y a trois types d'émetteurs-récepteurs: l'émetteur-récepteur à intrusion, celui-ci exige que le câble du réseau soit coupé; le vampire qui fait des trous dans le câble et le type interne qui est relié au câble par un connecteur en T.

Les types à intrusion et vampire sont utilisés pour les câbles Ethernet standards, et le type interne est utilisé pour le câble "*thinnet*". Avec le type à intrusion il est difficile de maintenir la séparation de 2.5 mètres entre les marques noires; dès lors, la totalité du réseau serait hors d'usage pendant l'installation et le câble deviendrait un ensemble de pièces de longueurs aléatoires.

Les types vampires sont un peu plus difficiles à installer, ils nécessitent des outils spécifiques suivant chaque marque. Le type Vampire ne dérange pas un réseau actif, et reste sur le câble quand on supprime l'émetteur-récepteur. Il a été adopté par presque tous les producteurs. Les installations "*thinnet*" utilisent habituellement les connexions en T BNC (*Bayo-Net-locking Connector for thin coaxial cable*) pour le montage directe des émetteurs-récepteurs internes sur l'ordinateur hôte.

Les émetteurs-récepteurs doivent être à au moins 2.5 mètres l'un de l'autre et doivent être placés sur les marques noires du câble Ethernet standard. Le nombre maximum d'émetteurs-récepteurs par segment est de 100; le maximum par Ethernet est de 1024.

Les diagrammes des différents types de branchements sont illustrés dans les figures ci-dessous:



I.1.1.2.4. Le câble de raccordement (*drop cable*)

Les câbles de raccordement connectent l'émetteur-récepteur, qui est déjà relié au câble du réseau, à la carte d'interface qui se trouve dans l'ordinateur hôte. Ces câbles sont disponibles chez la plupart des vendeurs de réseaux, ils peuvent être fabriqués avec des câbles à paire torsadée.

Comme le câble coaxial, le câble de raccordement a une couche extérieure qui peut être composée soit de Téflon soit de PVC. Les émetteurs-récepteurs Internet de "*thinnet*" n'utilisent pas un câble de raccordement mais se connectent directement sur le câble coaxial "*thinnet*".

I.1.1.2.4.1. Les connecteurs

Les câbles de raccordements utilisent des connecteurs mâles et femelles du type DB15: le connecteur femelle est destiné à l'extrémité de l'émetteur-récepteur et le connecteur mâle à l'extrémité de l'ordinateur. Les capuchons (*hoods*) métalliques sont utilisés pour protéger les fils du connecteurs et pour la protection extérieure.

I.1.1.2.4.2. L'affectation des épingles (*pins*)

Les câbles de raccordements ont des versions spécifiques et doivent faire correspondre les cartes d'interface avec les émetteurs-récepteurs. La version 1 de ce câble utilise une paire de fils de taille 20 et 3 paires de fils de taille 22.

Le picot n°1 est un fil de terre, ses parties protectrices intérieures et extérieures sont reliées ensemble au picot n°1 et au capuchon métallique. Les câbles de raccordements de la version 2 sont semblables à la version 1 excepté que toutes les 4 paires sont des fils de taille 20.

Les câbles IEEE 802.3 utilisent le picot n°4 comme fil de terre. Ils isolent les deux couches protectrices, relient la partie intérieure du câble au picot n°4 et la partie extérieure à la coquille métallique.

Quelques producteurs vendent des câbles appelés "câbles de raccordements de bureau", ce sont 4 paires de fils de taille 24 qui réduisent les distances des spécifications d'un facteur 4. Ils sont plus faciles et plus flexibles dans leur fonctionnement que les fils standards de taille 20, ils sont également moins chers mais leurs longueurs limitées sont rapidement dépassés.

Les spécifications Internet limitent les longueurs des câbles de raccordements: la longueur minimale est de 5 mètres; la distance maximale entre un émetteur-récepteur et un hôte est de 50 mètres, elle est de 40 mètres entre un émetteur-récepteur multi-port et un hôte.

Les câbles de raccordements de bureau ont une longueur totale limitée à 12 mètres et ne doivent pas être utilisés pour des multi-ports. Si les limites de longueurs des câbles de raccordements sont dépassées même d'un bit, les performances seraient dégradées, puisque le nombre de collisions et d'erreurs augmenteraient très rapidement. Si le dépassement est élevé le dispositif ne fonctionne plus. Les câbles en fibres optiques peuvent être d'une longueur de 5 mètres jusqu'à environ 5 km et forment la base des répéteurs optiques.

I.1.1.2.5. La connexion et l'expansion des réseaux Ethernet

Les segments Ethernet peuvent avoir des connexions logiques avec plusieurs points des 7 couches du modèle OSI.

Dans la couche 1 (couche physique) on utilise des connecteurs ou des répéteurs; les bits sont transférés par le matériel. Dans la couche 2 (liaisons de données), des ponts sont utilisés; les trames sont aussi transférées par le matériel seulement. A la couche 3 (couche réseau), des routeurs sont utilisés; tous les messages sont transférés, en utilisant à la fois le matériel et le logiciel.

Dans une topologie en bus, les extrémités sont "bouchées" par des petites pièces métalliques appelées terminateurs. Ces terminateurs ne sont, en fait, rien d'autre que des impédances de valeurs identiques à l'impédance caractéristique du câble. Toutes ces notions vont être développées plus loin.

Lorsqu'il faut agrandir le réseau, il suffit d'enlever un de ces "bouchons", de rajouter la nouvelle station de travail et de remettre le terminateur après celle-ci. Bien sûr, comme pour la plupart des réseaux, plus le nombre de stations connectées augmente, plus le temps de réponse augmente. La limite théorique est de 255 stations, mais cela reste bien sûr théorique car un réseau comportant un tel nombre de postes aurait une vitesse assez faible.

I.1.1.2.5.1. Les répéteurs

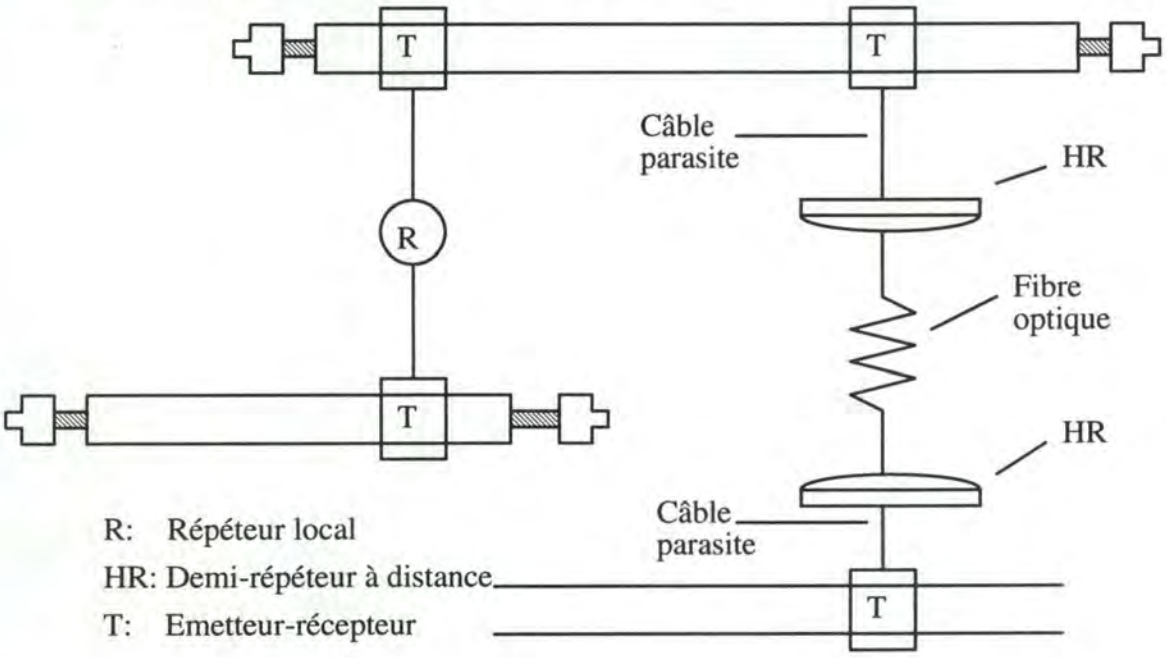
Un répéteur est un dispositif actif utilisé pour la connexion des segments Ethernet qui sont très éloignés, pour qu'ils soient reliés par des adaptateurs en tambours. Ils exigent une alimentation externe.

Un répéteur reconstitue les paquets, mais il ne les interprète pas; il ne connaît pas la destination d'un paquet ni le protocole utilisé. Les répéteurs locaux se trouvent entre les segments du réseau, ils sont reliés à chaque câble avec des câbles de raccordements et des émetteurs-récepteurs.

Les répéteurs distants se présentent en paires, une pour chaque côté de la connexion. Ils sont reliés aux segments du câbles avec des émetteurs-récepteurs et des câbles de raccordement. Ils sont également reliés l'un à l'autre par des câbles en fibre optique d'environ 1000 mètres.

La figure ci-après illustre des connexions de répéteurs locaux et distants:

Les connexions des répéteurs à distance et locaux:



Les répéteurs sont souvent utilisés pour connecter des segments d'Ethernet dans un bâtiment ou dans des bâtiments proches.

Les versions 1 et 2 d'Ethernet spécifient au plus 2 répéteurs en série par réseau; IEEE 802.3 a étendu la spécification à 4 répéteurs.

Les répéteurs à distance nécessitent l'assistance d'un administrateur du système, pour éviter par exemple des problèmes d'accès.

I.1.1.2.5.2. Les ponts (bridges)

Les ponts connectent les câbles Ethernet au niveau liaison de données (couche 2) du modèle ISO. Ils n'exigent pas un logiciel, mais reçoivent, régénèrent, et retransmettent les paquets vers le matériel.

Plusieurs ponts ont un algorithme dynamique d'apprentissage: le pont note les adresses sources qu'il reçoit de son côté gauche et de son côté droit et transmet les paquets au côté opposé seulement si le hôte de destination se trouve de l'autre côté.

Au début, les paquets sont transmis, mais après quelques minutes toutes les adresses des hôtes qui ont transmis les paquets sont mémorisées par le pont. Les ponts sont des protocoles indépendants et peuvent manipuler tous types de paquets.

Les ponts doivent explorer chaque paquet pour déterminer s'ils doivent les transmettre. Leur performance est d'habitude mesurée par le taux d'exploration des paquets et par le taux de leur transmission.

Quand un réseau est composé de plusieurs boucles les ponts deviennent plus confus, puisque les paquets transmis par un même hôte apparaissent au 2 côtés du pont. Un seul réseau Ethernet ne peut pas avoir des boucles. Toutefois, si l'on connecte plusieurs Ethernet avec des routeurs ou des ponts, la typologie peut inclure plusieurs chemins vers un hôte.

Quelques ponts peuvent résoudre cet inconvénient en mémorisant les différentes routes qui forment la boucle au cas où cette route primaire tombe en panne.

D'autres ponts peuvent également manipuler la duplication des liaisons entre les 2 mêmes réseaux et routent le trafic à la manière "*round-robin*".

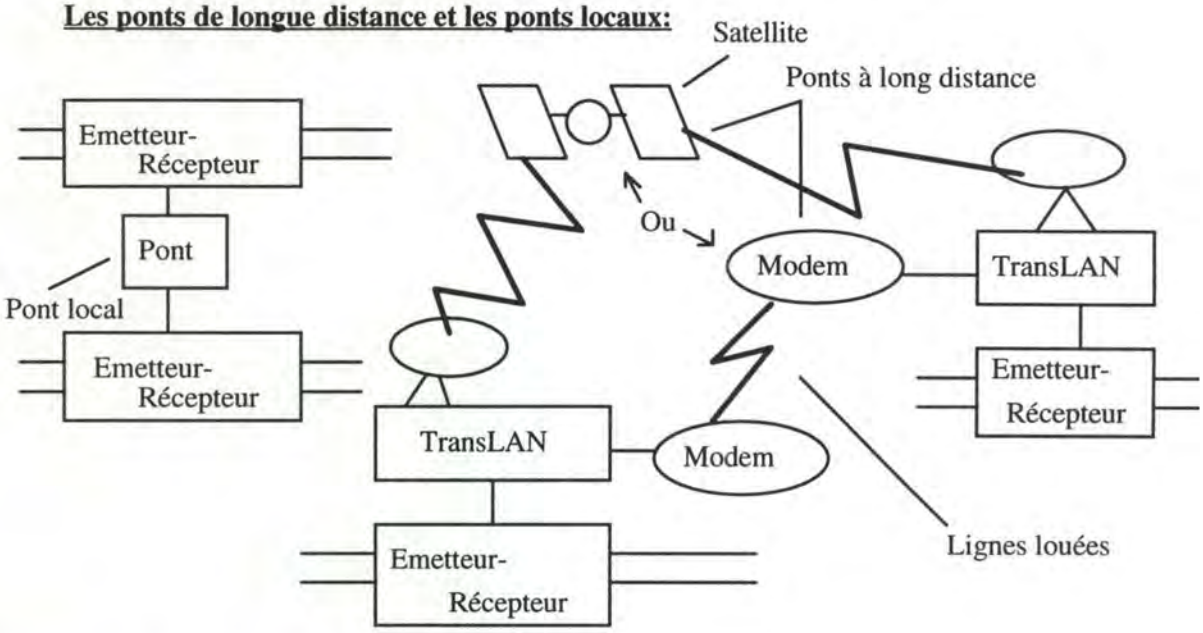
Le pont *transLAN* peut connecter 2 Ethernets éloignés via un système de communication digital tel que le satellite ou des lignes louées nationales. La largeur de bande du système actuel est de 56 Kbps ou 1.54 Mbps (T1 est la terminologie des compagnies de téléphone pour les canaux de 1.54 Mbps).

Le *transLAN* fonctionne comme un pont, mais comme les liaisons satellites sont plus lentes, l'utilisateur, travaillant de façon interactive, peut remarquer des retards. Dans un site local il utilise une interface Ethernet et au niveau du satellite ou du site des lignes louées il utilise V.35 ou le standard RS-232 .

Les *transLANs* sont programmés par la console elle-même et contiennent un logiciel de gestion qui permet à l'utilisateur de faire la configuration du matériel pour le contrôle du filtrage des paquets. Ils sont également transparent aux protocoles de haut niveau.

La configuration du réseau des ponts locaux et *transLAN* est illustrée dans la figure ci-dessous. Le budget de temps Ethernet limite les réseaux à un maximum de 7 ponts en série

Les ponts de longue distance et les ponts locaux:



I.1.1.2.5.3. Les routeurs

Les routeurs sont des ordinateurs qui contiennent deux ou plusieurs interfaces Ethernet et qui fonctionnent comme un agent de circulation dans un carrefour encombré.

Un ordinateur principal est souvent utilisé comme routeur pour des raisons économiques. On utilise les serveurs des fichiers comme routeurs pour des réseaux ayant des stations de travail sans disques puisque la grande partie du trafic dans le réseau local est entre le serveur et ses clients; ainsi le chargement supplémentaire laissé chez le serveur fonctionnant comme un routeur n'est plus important.

I.1.1.2.5.4. Les passerelles (*Gateways*)

Les passerelles sont des machines hôtes qui font la conversion des protocoles, elles s'occupent également du routage. Ainsi, une machine qui parle à la fois Ethernet et X.25 et qui passe des paquets entre ces deux réseaux sera appelé passerelle. Les routeurs sont souvent appelés passerelles, surtout s'ils sont des ordinateurs principaux très chargés.

Un autre type de passerelle est celui du courrier qui doit faire la conversion nécessaire pour permettre aux messages destinés à un hôte, par exemple, IBM BITNET d'arriver sur un Internet TCP/IP et être reroutés et remballés en une forme comprise par le système BITNET.

I.1.1.3. Les normes

Ethernet est normalisé sous la norme IEEE 802 (ISO 8802) qui est consacrée aux réseaux locaux (802.4, Bus à jeton; 802.5, anneau à jeton; 802.3, méthode d'accès CSMA/CA OU CSMA/CD).

I.1.1.4. Les échanges

Le robinet applique des connexions électriques sur les deux conducteurs du câble coaxial du réseau: le conducteur central et le conducteur de masse. L'émetteur-récepteur est attaché au robinet, il transmet les signaux via le câble de raccordement vers la carte d'interface Ethernet qui se trouve sur l'ordinateur hôte.

Lorsqu'un message est émis par une station, il est transmis dans les deux sens à toutes les stations qui doivent alors déterminer si le message leur est destiné. Les stations de travail sont dites passives car elles ne doivent pas réamplifier le signal, comme ce serait le cas sur un réseau en anneau.

Cette particularité présente un double avantage:
- le temps que l'information passe dans chaque noeud, on gagnera ici en vitesse;
- si l'une des stations de travail est défectueuse, elle ne risque pas de paralyser tout le réseau puisque chaque poste ne joue qu'un rôle passif.

I.1.1.5. Les limites et les tailles d'Ethernet

Quand on augmente le nombre des machines dans un réseau donné, celui-ci devient moins fiable et plus lent dans son fonctionnement. Ce problème est relié aux limitations des distances et au nombre maximum de composants pour la construction d'un Ethernet. Les standards d'Ethernet se chargent de tous ces spécifications.
Les limites et les tailles d'Ethernet sont résumés dans le tableau ci-dessous :

Tailles et limites d'Ethernet:

Composants	Limites
Sections *"thicknet"* Segments *"thicknet"* Sections *"thinnet"*	Longueurs standards: 23.4, 70.2 et 117 mètres Long de 10 à 500 mètres Longueur maximale: 185 mètres
Emetteur-Récepteurs	Sur les marques noires du *"thicknet"* Séparés de 2.5 à 1500 mètres Au plus 100 par segment Au plus 1023 par Ethernet Au moins séparés de .5 mètres pour *"thinnet"*
Câble de raccordement Câble de raccordement de bureau	De 5 à 50 mètres de longueur pour des*"transceivers"* discréts De 5 à 40 mètres de longueur pour des*"transceivers"* multi-ports De 5 à 5000 mètres pour la fibre optique De 2 à 12.5 mètres de longueur pour les discréts
Répéteurs	Version 1: maximum de deux en série Version 2 et IEEE 802.3: un maximum de 4 en série De 10 à 100 mètres entre les segments pour les répéteurs locaux De 10 à 1100 mètres entre les segments pour les répéteurs à distance De 10 à 3000 mètres entre les segments pour les répéteurs optiques
Ponts	Un maximum de 7 en série

I.1.1.6. Conclusion

De nos jours, la plupart des systèmes de câblage Ethernet utilisent des paires torsadées non protégées (ou UTP pour U*nshielded Twisted Pair*), ou des câbles coaxiaux (*thin* ou *thick)* pour transporter les paquets dans le LAN. La paire torsadée est composée de deux paires de fils tordus. La paire torsadée est utilisée dans la topologie en étoile pour les réseaux Ethernet du type 10Base-T. Les UTP parcourent le réseau à partir des noeuds jusqu'au centre, ce dernier être connecté à d'autres parties du réseau via un *backbone.*

On tord les fils pour réduire les interférences électriques. La protection se fait en ajoutant une isolation autour du fil, ainsi, on l'immunise contre le bruit. Les laboratoires distinguent trois catégories pour les câbles UTP: la catégorie 1 (pour la voix; appelé aussi satin argenté), la catégorie 3 (pour les données), ou la catégorie 5 (données à vitesse élevée). Un réseau 10Base-T necessite la catégorie 3, alors que la catégorie 5 est utilisée dans les nouveaux systèmes Ethernet de 100Mbps.

La paire torsadée protégée (ou STP pour *Shielded Twisted Pair*) utilisée pour quelques réseaux *Token Ring* ressemble aux fils électriques ménagers. STP transporte un signal relativement de bas voltage, il est entouré d'une couche d'isolation pour réduire les bruits, il est cependant risqué.

Le câble coaxial est plus cher que la paire torsadée, il présente plus d'avantages. Les câbles coaxiaux du type *Thick* permettent de parcourir des distances plus grandes et permettent également d'attacher plusieurs noeuds, car il est moins prédisposé au signal d'interférence et d'atténuation (Pour cette raison, le câble coaxial est également utilisé pour la distribution TV). Les sections des réseaux Ethernet utilisant le câble coaxial employent une topologie linéaire de type bus.

Le câble coaxial standard, ou le *Thicknet,* a un degré élevé d'immunité contre le bruit, il est plus difficile à détruire, mais il éxige une combinaison d'un "robinet" de type "Vampire" et un câble de raccordement pour se connecter sur le LAN. Les nouveaux câbles *Thinnet* sont plus fins que le *Thicknet.* Malgré que le *Thinnet* ne peut pas transporter les signaux sur des distances aussi longues que celle du *Thicknet,* il utilise des connecteurs simples du type BNC (pour *Bayo-Net-locking Connector for thin coaxial cables*), il coûte moins cher et est devenu un standard pour les LAN Ethernet de petite ou de moyenne taille.

Dans un réseau Ethernet, le nombre de connexions et leurs distances peuvent constituer des facteurs de limitation pour un type particulier de câble. Pour le *Thicknet,* on peut utiliser des répéteurs pour regénérer le signal, et ce tous les 500 métres environ. Les ondes permanentes (réflexion supplémentaire des signaux) déforment le signal et par conséquent causent des erreurs dans un long réseau sans répéteurs. La détection des collisions (quand deux adaptateurs de réseaux essayent de transmettre en même temps) dépond partiellement du *timing*; seulement cinq segments de 500 métres et quatre répéteurs peuvent être placés en série pour que le temps de propagation du signal devienne plus long que la période de temps maximale permise pour la détection d'une collision. Sans cette limite, la station de travail la plus éloignée de l'émetteur serait incapable de déterminer si une collision a eu lieu.

I.1.2.UNIX

I.1.2.1. Présentation d'Unix

I.1.2.1.1. Historique

Unix est devenu de nos jours un des systèmes d'exploitation le plus populaire. Vu l'efficacité et les performances de ses commandes, Unix a pu attirer tous les niveaux d'utilisateurs. Parmi les différentes versions d'Unix utilisées, les systèmes développés par une équipe de l'université de Californie, Berkeley, au début des années 80. L'Unix de Berkeley est de plus en plus sollicité surtout dans les milieux universitaires, tant au niveau de l'éducation qu'à celui de la recherche, et s'est également élargit au domaine publique, au gouvernement et à l'industrie.

Unix a été développé dans les laboratoires Bell, une filiale de l'AT&T, dans les années soixante. En 1966, les laboratoires Bell ont eut besoin, pour leur usage interne, d'un système d'exploitation multitâche adapté au traitement de textes et au développement d'applications. Une équipe dirigée par Ken Thomson s'est occupée de ce travail. En 1969, le projet aboutit à la première version d'Unix.

Il devient ainsi rapidement nécessaire de porter le système Unix sur d'autres ordinateurs. En 1973, Denis Ritchie réécrit entièrement Unix en C, le rendant ainsi portable.

Dès 1974, AT&T propose les premières licences aux universités. La masse logicielle d'Unix s'enrichit d'extensions et d'utilitaires variés. L'université de Berkeley apporte à Unix de nombreuses améliorations.

En 1978, AT&T présente aux industriels les premières versions commerciales d'Unix : Unix V32 pour les ordinateurs VAX 11/780 et Unix V7 pour les DEC PDP11.

I.1.2.1.2. Versions récentes d'Unix

Dans les années 80, AT&T autorise le clonage des systèmes par d'autres constructeurs. Plusieurs versions de Unix naissent, mais à la base elles sont toutes issues du même système qui est adapté à tourner sur des machines différentes. En effet, la majorité des constructeurs d'ordinateurs offrent, de nos jours, le système Unix avec leurs propres améliorations: ULTRIX pour Digital Equipment Corporation (DEC), BSD pour SUN, HP-UX pour Hewlett-Packard, Domain pour APOLLO et AIX pour l'International Business Machines (IBM), La société Microsoft ouvre le marché d'Unix aux micro-ordinateurs en proposant le système Xenix pour l'IBM PC.

Les deux versions d'Unix les plus utilisées pendant les années 80, sont principalement: Unix Système V commercialisé par l'AT&T et BSD 4.2 issu de l'UCB (Université de Californie, Berkeley). Ces versions, BSD (Berkeley Software Distribution) et système V, ont connu

plusieurs améliorations, mais l'UCB a dépassé l'AT&T en ajoutant un nouvel éditeur de texte, une nouvelle interface utilisateur et surtout, une gestion de mémoire virtuelle, ce qui a rendu le système idéal pour les ordinateur VAX à 32-bits introduits par DEC à la fin des années 70.

L'ajout de la mémoire virtuelle et la politique mené par l'UCB qui consiste aux remises accordées sur les licences d'utilisation de Unix par les organismes d'éducation, a causé une augmentation accrue de l'utilisation du système dans les collèges et les universités, commençant par le 4.1 BSD en 1981 et continuant jusqu'à la récente version, 4.3 BSD. AT&T, à son tour, a incorporé quelques unes des meilleurs caractéristiques de Berkeley pour sa dernière version Unix système V en 1983.

Aujourd'hui Unix est devenu le standard de l'industrie, c-à-d le plus utilisé. Sur des PC, les stations de travail, les mini-ordinateurs, et même sur les plus puissants des "mainframes", Unix fournit un environnement informatique efficace, simple, et élégant pour une productivité sans cesse croissante.

Unix système V est surtout utilisé dans des moyennes ou petites configurations pour des applications informatiques traditionnelles. Les améliorations qui ont été apportées par rapport à la version III sont:

- la gestion des fichiers: les blocs disques ont augmenté de volume, passant de 512 à 1024 octets;
- le noyau: la nouvelle taille des blocs entraîne une réduction des entrées/sorties. Cela réduit fortement le temps de traitement des tâches;
- les utilitaires: tant l'éditeur pleine page "vi" que la possibilité de faire communiquer entre eux des systèmes Unix sont disponibles;
- l'interpréteur de commandes (le shell) s'oriente vers un langage de programmation procédurale .

La version BSD 4.2 se retrouve souvent sur des configurations importantes supportant des applications scientifiques. Les caractéristiques nouvelles apparues sur la version 4.2 sont les suivantes:

- un nouveau mécanisme de communication entre processus;
- une nouvelle organisation du système de fichiers;
- de nouveaux utilitaires: compilateur Fortran, courrier électronique, et le shell réparti (remote shell);
- un protocole réseau Arpanet.

I.1.2.1.3. Fonctionnalités

Le système Unix comprend trois grandes fonctionnalités:

- une fonctionnalité qui consiste à gérer les ressources de l'ordinateur. Unix étant multitâches et multi-utilisateurs, plusieurs utilisateurs peuvent demander l'exécution d'une ou de plusieurs tâches simultanément. Le noyau d'Unix répartit les différentes ressources

de l'ordinateur entre les tâches et entre les utilisateurs de façon transparente pour ces derniers;

- Unix propose deux autres fonctionnalités qui consistent en:
  1. un système de gestion de données pour l'organisation, la maintenance et l'accès aux unités de stockage;
  2. un système de communication entre utilisateurs;

- Unix possède un environnement de programmation très complet comprenant entre autres un compilateur C et Fortran, des éditeurs, des traitements de textes et de nombreux outils d'aide à la programmation: analyseurs syntaxiques, débogueurs, générateurs de sources, etc.

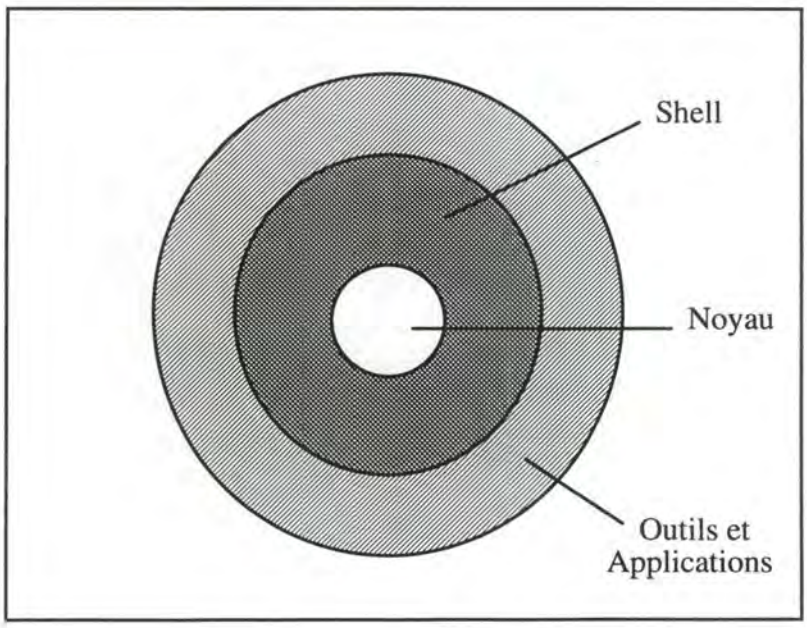
I.1.2.2. La structure du système Unix

Les parties du système Unix peuvent être fonctionnellement divisés en trois niveau: le noyau, le shell, et les applications et outils:

- le noyau (*kernel*) programme les tâches et gère le rangement des données;
- le shell est un programme qui connecte et interprète les commandes introduites par l'utilisateur. Il appelle les programmes de la mémoire, et les exécute individuellement ou en série (*pipe*);
- les outils et les applications apportent des capacités supplémentaires au système d'exploitation.

Le schéma ci-dessous illustre la structure de ces niveaux:

Les composants du système UNIX:



I.1.2.2.1. Le Noyau

Le Noyau est le coeur du système d'exploitation. Il commande et contrôle le matériel. Il fait fonctionner des parties du système de l'ordinateur à partir de programmes de commande. Tous les systèmes d'exploitation ont un noyau qui peut parfois avoir un nom différent. En tapant, par exemple, la commande **ls** (pour lister les noms des fichiers et sous-répertoires contenus dans le répertoire courant), le noyau va diriger l'ordinateur d'abord pour lire les noms des fichiers sur le disque et ensuite les afficher sur l'écran pour l'utilisateur.

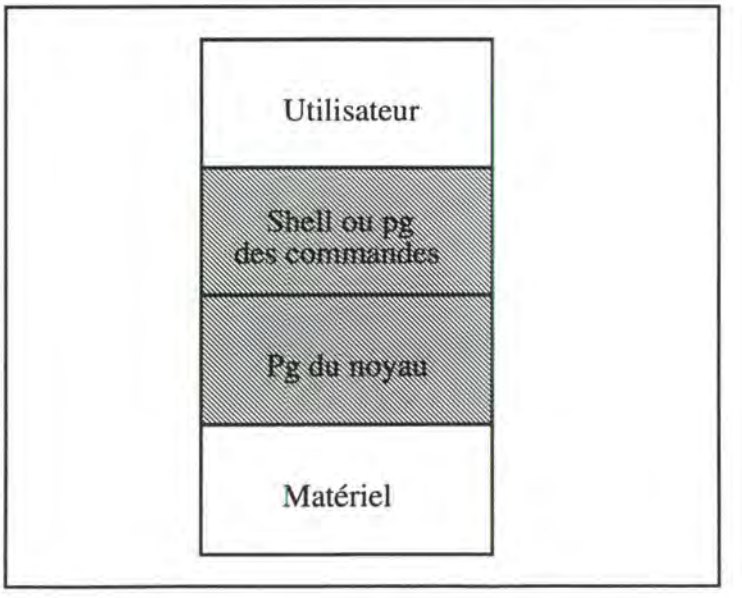
Le programme noyau gère les ressources de l'ordinateur. Ces ressources sont: la mémoire de l'ordinateur, le disque, les lecteurs de bandes ou de disques, les terminaux, les imprimantes, et les autres composants du matériel.

Le système de fichiers qui réside sur le disque et qui fournit une organisation à toutes les données du système Unix est également considéré comme ressource et est géré directement par le noyau. Le programme noyau est toujours résident dans la mémoire de l'ordinateur de sorte qu'il est prêt pour une exécution avec un minimum de retard.

Les utilisateurs d'un système Unix ne traitent pas directement avec le programme noyau mais avec un programme intermédiaire, qui peut être un shell ou un programme de commande. Ce programme intermédiaire accède au programme noyau, ainsi, l'utilisateur voit un shell ou un programme de commande quand il regarde de haut le système Unix. Le noyau voit également un shell ou un programme de commande quand il regarde d'en bas vers l'utilisateur.

La figure ci-dessous illustre ces relations avec le modèle en couches:

Les couches du logiciel du système UNIX:



Le programme de commande peut être un programme utilitaire Unix, ou un programme interactif complexe.

I.1.2.2.2. Le shell

Le shell est un interprèteur de commandes bâti autour du noyau, il permet la connexion entre l'utilisateur et l'ordinateur. Comme un interpréteur humain qui se met entre deux personnes qui parlent des langues différentes et traduit la langue de l'un d'entre eux dans celle de l'autre. Le shell, quant à lui, se met entre l'utilisateur et le noyau, il parle à la fois la langue de l'utilisateur et celle de la machine. Le programme shell interprète les commandes qui sont tapées par l'utilisateur et les traduit en commandes comprises par le noyau. Il demande au kernel d'exécuter le travail que l'utilisateur a demandé en éliminant ainsi le besoin de parler avec le kernel.

Il y a plusieurs shell disponible pour les utilisateurs Unix. Le shell de Bourne et le C shell sont les plus rencontrés sur le système Unix. D'autres shells (menus) sont d'habitude vendus comme des progiciels. Les shells "menus" sont faciles à utiliser, plusieurs utilisateurs Unix prennent les avantages des différents shells pour des jobs différents. Les utilisateurs débutants utilisent les shells "menu" qui sont les plus faciles alors que les utilisateurs les plus expérimentés préfèrent utiliser C shell ou le shell de Bourne.

Le shell offre également des moyens de chaînage de commandes (*pipelining*) à l'aide des tubes (*pipelines*). Pour entrer dans un système de tubes, une commande doit pouvoir accepter des données externes (sauf si elle est située au début du tube) et fournir des données récupérables.

Par exemple, les commandes **mkdir, cd, rm** ne produisent aucune donnée: elles interviennent dans le système. Les commandes **date, who, ls,** quant à elles, n'acceptent aucune donnée en entrée: elles vont chercher leurs informations dans le système.

Le shell trouve le programme demandé par la commande de l'utilisateur, le transfère dans la mémoire, et l'exécute. Quand le shell exécute cette procédure, on dit qu'il évoque un programme. Quand la commande est terminée, le "prompt" du shell réapparaît, indiquant que le shell est prêt à répondre à une nouvelle commande.

Si le shell ne peut pas localiser le programme de commande spécifié par l'utilisateur, il envoie simplement un message d'avertissement et affiche un autre prompt, indiquant que l'utilisateur peut essayer des nouvelles commandes.

Un programme en exécution est connu sous le nom de processus. Ainsi, un shell en exécution est parfois appelé un processus shell , et un programme de commande en exécution, un processus de commande. Le système Unix est un système d'exploitation multitâche, ce qui veut dire qu'on peut programmer l'exécution de plus d'un processus ou tâche à la fois. Si le système a seulement un seul CPU, alors un seul processus peut être exécuté à un instant donné. Cependant, le noyau est capable de manipuler les processus de façon à suspendre l'exécution de l'un pour passer à l'exécution de l'autre et ensuite revenir au point de suspension du premier. Et comme le système Unix est capable de passer d'un processus à un autre très rapidement, tous les processus semblent tourner simultanément, même si le temps CPU est en réalité partagé. Pour cette raison, Unix est considéré comme un système à temps partagé.

Une importante caractéristique des shells Unix est qu'ils sont interchangeables; ce qui veut dire qu'on peut évoquer un shell à partir d'un autre simplement en tapant le nom du shell

désiré comme n'importe quelle commande. Ceci est possible car l'interpréteur de commandes (le shell) ne fait pas partie du programme noyau; le shell s'exécute plutôt indépendamment du programme noyau comme n'importe quel autre programme de commande.

Le shell offre un environnement pour l'exécution des programmes de commande, et comme un programmeur peut écrire un nouveau programme de commande, l'environnement dans lequel les commandes s'exécutent peut être facilement changé en changeant les shells.

I.1.2.2.3. Les outils Unix

La troisième couche (la plus extérieure) du système contient les outils Unix. Ces outils varient d'une implémentation à l'autre. Quelques implémentations comportent plus ou moins 400 outils, d'autres comportent seulement un sous-ensemble approprié à un certain type d'utilisateur (utilisateur de traitement de texte, programmeur, etc.).

I.1.2.3. Les utilisateurs

Tout utilisateur reçoit de l'administrateur du système un **nom**, qui l'identifie par rapport aux autres utilisateurs, ainsi qu'un **répertoire personnel** destiné à enregistrer tous ses fichiers. Tout utilisateur reçoit certains privilèges, restreints, comme celui de pouvoir lire et écrire dans son répertoire propre.

Le répertoire courant est le répertoire dans lequel on se trouve à un moment donné; on l'appelle aussi le répertoire de travail. Lorsque l'utilisateur se connecte au système, il a automatiquement pour répertoire courant son répertoire personnel, appelé aussi **$HOME.**

Mais un utilisateur fait également partie d'un ou plusieurs **groupes** d'utilisateurs. Chaque groupe possède certains privilèges, comme celui de pouvoir exécuter certaines applications. C'est l'administrateur du système qui place un utilisateur dans un groupe dans lequel il partage les privilèges.

Tous les renseignements relatifs à un utilisateur (le nom, le répertoire personnel et les groupes dont il fait partie) sont regroupés dans ce que nous appellerons un **compte**, créé par l'administrateur du système.

I.1.2.3.1. Accès au système

L'accès au système (ou *login*) est la première opération à effectuer pour dialoguer avec le système Unix. Cette opération d'identification consiste à donner un nom suivi, éventuellement, d'un mot de passe. L'utilisateur doit répondre aux messages lui demandant de s'identifier.

Pour des raisons de confidentialité et de sécurité, les caractères composant le mot de passe (*password*) ne sont pas affichés à l'écran. Ils sont généralement remplacés par des pavés graphiques ou des x, voire même des espaces.

Lorsque le *login* est correct, Unix initialise le contexte de l'utilisateur. Il affiche ensuite un message d'accueil (*prompt*) indiquant à l'utilisateur qu'il peut entrer ses commandes. Si le *login* n'est pas correct le système relance la procédure d'identification.

Il est important de noter que:

- l'utilisateur (associé à un mot de passe) est reconnu par le système lorsque l'administrateur du site l'a identifié à l'intérieur du fichier /etc/passwd.
- l'administrateur inscrit aussi l'utilisateur à l'intérieur d'un groupe d'utilisateurs. L'appartenance à un groupe est utilisée pour la confidentialité des données.

I.1.2.3.2. Sortie du système

Selon le système sur lequel on était branché et le shell actif, il existe trois manières de sortir:

- entrer la commande **exit** et appuyer sur <Entrée>;
- entrer la commande **logout** et appuyer sur <Entrée>;
- appuyer sur <Ctrl>+<d>.

La commande devra éventuellement être exécutée plusieurs fois, par exemple dans le cas où on travaille dans un sous-shell.

Le shell reçoit un signal de terminaison. L'utilisateur est déconnecté du système. Le système se met alors en attente d'une autre connexion.

I.1.2.3.3. Contexte

Le contexte d'un utilisateur correspond à un ensemble de variables systèmes (ou définies par l'utilisateur). Ces variables sont initialisées lors du login.

Après s'être connecté au système, l'utilisateur se trouve dans un répertoire particulier de l'arborescence Unix. Dans ce répertoire se trouve un fichier de commandes, exécuté après le login et définissant le contexte de travail de l'utilisateur. Ce fichier se nomme **.profile** dans le shell Bourne et **.login** dans le C shell.

Dans le fichier .profile (ou .login) est initialisé un ensemble de variables parmi lesquelles:

- **HOME:** (ou home) définit le répertoire de travail de l'utilisateur. C'est aussi le répertoire où il se trouve juste après le login;

- **TERM:** (ou term) définit le type de terminal utilisé;

- **PATH:** (ou path) définit une liste de chemins d'accès vers des fichiers exécutables. Si le système ne trouve pas un programme sous le répertoire courant, il le cherche dans la variable PATH. Chaque élément de la liste est séparé du suivant par le caractère ":" en shell Bourne. En C shell, la liste des chemins d'accès est mise entre parenthèses et chaque élément est séparé du suivant par le caractère <espace>;

- **CDPATH:** (ou cwd) définit la liste des nouveaux répertoires (répertoires où aller chercher un répertoire) lors de l'utilisation de la commande *change directory*.

- **PS1:** (ou prompt) définit le message indiquant que le système est en attente d'une commande;

- **PS2:** définit le message indiquant que le shell Bourne attend encore d'autres informations;

- **MAIL:** (ou mail) définit le nom du fichier servant de boîte aux lettres pour l'utilisateur;

- **shell:** définit dans le C shell le chemin, dans l'arborescence Unix, vers le programme C shell (csh).

Il est possible d'inclure dans le fichier ".profile" (ou ".login") des commandes lancées automatiquement lors du login.

I.1.2.4. Organisation des fichiers Unix

I.1.2.4.1. Arborescence d'Unix

Sous Unix, les répertoires sont considérés comme des fichiers. Chaque fichier possède un nom composé de 14 caractères. Contrairement au système MS-DOS, le caractère "." n'a pas de signification particulière. Il sert juste de délimiteur pour l'extention. D'autre part, le système de gestion de fichiers distingue les majuscules des minuscules à l'intérieur des noms de fichiers. Cette distinction est typique du langage C. Le répertoire racine se nomme "/", c'est le répertoire se trouvant au plus haut niveau de l'arborescence Unix.

Dans le système Unix (comme sous MS-DOS) chaque répertoire possède deux sous-répertoires particuliers:

- le répertoire "." ou répertoire courant. C'est l'endroit où l'on se trouve dans l'arborescence. Le nom de ce répertoir est visualisé en utilisant la commande **pwd**;

- le répertoire "..", répertoire père du répertoire courant.

Il existe deux méthodes pour désigner un fichier: le nommer par son nom absolu ou par son nom relatif:
- le nom absolu d'un fichier représente le chemin, dans l'arborescence, depuis la racine jusqu'au fichier concerné. Chaque nom de répertoire intermédiaire est séparé par le caractère "/" ("\" sous MS-DOS);
- le nom relatif est construit de la même manière, mais il représente le chemin depuis le répertoire courant.

L'extension d'un fichier révèle souvent la nature de son contenu. Par exemple:

- un fichier sans extension est le plus souvent un fichier exécutable (fichier .EXE ou .COM sous MS-DOS) ou un répertoire;

- .a: fichier d'archive. C'est une bibliothèque de fonctions à laquelle on accède en l'incluant à un programme lors de l'édition des liens. Par exemple, le fichier "libc.a" est la bibliothèque de fonctions utilisée par défaut par le compilateur C;
- .c: fichier source en langage C;
- .h: fichier d'en-tête. Ces fichiers sont inclus dans les sources des programmes C et Fortran pour déclarer les fonctions externes utilisées. Le corps de ces fonctions est inclus à l'édition de liens;
- .o: fichier objet;
- .profile: fichier particulier se trouvant au niveau de chaque répertoire utilisateur. Ce fichier définit le contexte d'utilisation. Il est lu lors de l'entrée d'un utilisateur dans le système.

La commande **file** donne des indications sur la nature des fichiers.

I.1.2.4.2. Attribut des fichiers

Chaque fichier possède un ensemble d'attributs définissant les droits pour tous les utilisateurs du système. Il existe trois niveaux de confidentialité dans le système Unix:

- le propriétaire;
- le groupe;
- les autres.

A sa création, un fichier appartient à son créateur et fait partie du groupe de celui-ci. Le propriétaire d'un fichier peut distribuer ou restreindre les droits sur ce fichier. Ainsi, pour chaque niveau de confidentialité, le propriétaire donne ou restreint les droits d'accès à ses fichiers. Ces droits concernent la lecture, l'écriture et ou l'exécution de fichiers.

Le propriétaire d'un fichier et l'administrateur sont les seules personnes autorisées à modifier les modes d'un fichier. La commande **chmod** permet de modifier les modes d'accès associés à un fichier.

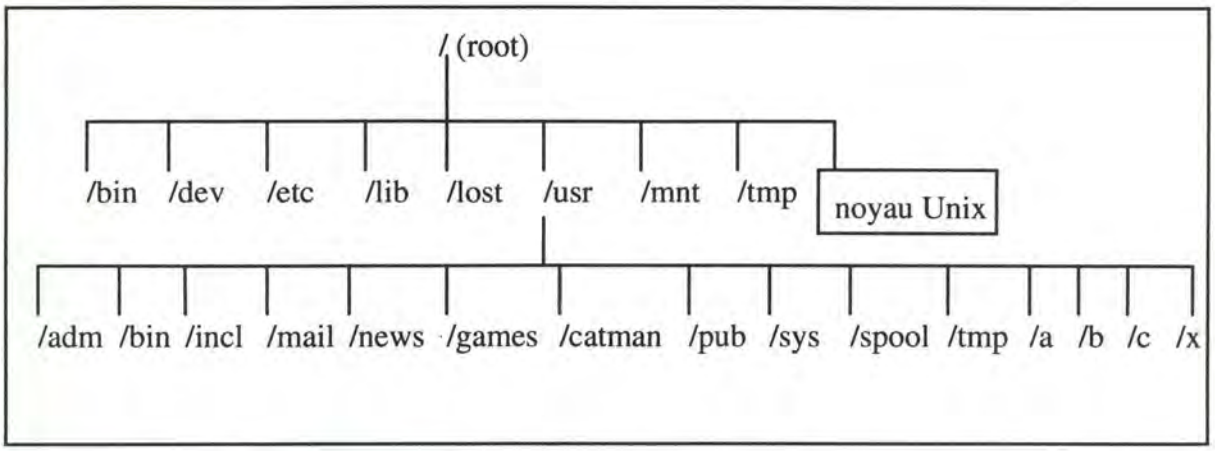
I.1.2.4.3. Organisation traditionnelle

Le système de fichier Unix est semblable aux branches d'un arbre renversé. Au sommet, il y a la racine d'où partent plusieurs branches (répertoires et/ou fichiers) et de chaque branche plusieurs feuilles (fichiers et/ou répertoires), comme le montre le schéma ci-dessous, le niveau initial des répertoires est standard à la plupart des systèmes Unix.

Chacun de ces répertoires contient une fonction majeure du système d'exploitation. Dans un répertoire nous pouvons établir un système de fichier efficace, nous pouvons diviser l'espace de travail en plusieurs sous-répertoires et déplacer des données dans ces sous-répertoires selon les besoins.

Le schéma ci-dessous illustre ce système de fichiers:

Le système de fichiers UNIX:



Le système Unix est organisé en niveaux hiérarchiques. Le premier niveau est réservé à la racine (*root*) de l'arborescence Unix. Lorsque l'utilisateur se trouve sous le root, le symbole "/" est affiché.

Le deuxième niveau contient les informations relatives au système en général; imprimantes, terminaux, drivers, commandes, etc. Ces informations sont organisées dans les répertoires suivants:

- **/bin:** regroupe les informations accessibles au le shell. Il s'agit de commandes système comme la commande **ls** ou **pwd**.

- **/dev:** (*device*) contient les drivers spécifiques au matériel ainsi que leurs configurations : connexion Transpac, imprimantes, terminaux, etc.

- **/etc:** (et caetera) contient les commandes réservées au super-utilisateur (généralement l'administration du système). Le fichier des mots de passe et la commande **fsck**, réalisant un diagnostic du système, se trouvent dans ce répertoire.

- **/lib:** (*library*) est la bibliothèque système contenant tous les modules nécessaires à l'édition de liens après une compilations (en particulier pour la programmation en C).

- **/lost:** permet de retrouver les fichiers dont Unix a perdu la trace. Ce répertoire contient les fichiers localisés mais ne possédant pas de référence ainsi que des informations sur des fichiers effacés.

- **/usr:** (*users*) concerne les applications et quelques commandes des utilisateurs ainsi que les comptes des utilisateurs.

- **/mnt:** est le répertoire où est habituellement lancée la commande **mount**, dont le rôle consiste à installer les périphériques dans le système de fichiers.

- **/tmp:** (*temporary*) regroupe les fichiers temporaires écrits lors d'une compilation. Ces fichiers sont effacés après la constitution des fichiers définitifs. Le répertoire tmp est généralement nettoyé après une initialisation du système (*reboot*).

Les répertoires contenant les informations relatives aux utilisateurs se situent au niveau trois:

- **/usr/adm:** (*administrator*) regroupe les commandes réservées à l'administrateur et concernant la maintenance du système. Le fichier /usr/adm/wtmp contient des infomations sur chaque utilisateur actif dans le système. Ces données sont accessibles par les commandes **who** ou **login**.

- **/usr/bin:** contient d'autres commandes accessibles par le shell ainsi que les applications du système (traitement de texte, base de données, etc.). Il s'agit de commandes moins fréquemment utilisées que celles se trouvant sous /bin.

- **/usr/incl:** (*include*) contient tous les fichiers de définition qui concernent le langage C. Ceux-ci sont inclus dans un programme par une instruction C du type: #include <nom-fichier.h>.

- **/usr/mail:** contient le courrier électronique.

- **/usr/news:** regroupe les informations générales entrées par l'administrateur à l'attention des utilisateurs du système.

- **/usr/games:** comprend un ensemble de jeux, dont certains sont multi-utilisateurs.

- **/usr/catman:** (*manual*) est le répertoire contenant la documentation du système. Celle-ci est affichée à l'aide de la commande **man**.

- **/usr/pub:** (*public*) regroupe les fichiers destinés à être lus (comme les fichiers mail ou news) ou exécutés par l'ensemble des utilisateurs.

- **/usr/sys:** (*system*) contient les sources du noyau Unix.

- **/usr/spool:** (*spooler*), sorte de mémoire tampon intelligente qui contient des fichiers et une table de références gérant des priorités (exemple: **lp** pour imprimantes et **uucp** pour réseau commuté).

- **/usr/a,b,c,...x:** sont les noms des utilisateurs. Ils constituent leurs répertoires personnels.

La hiérarchie du système de fichiers est décrite par la commande: **#man hier**.

I.1.2.5. Les processus

I.1.2.5.1. Notion de processus

Toutes les tâches des utilisateurs sont réalisées par des processus. Un processus est une séquence d'instructions susceptible d'être exécutée. Chaque processus est caractérisé par son environnement: le code, les données temporaires et permanentes, les fichiers et variables associés au programme.

Le système enrichit l'environnement d'un processus en lui attribuant les descripteurs, la mémoire allouée et la pile d'exécution du noyau. On appelle image d'un processus l'environnement d'exécution de celui-ci.

Pour chaque processus, il existe trois fichiers particuliers permettant de dialoguer avec l'environnement extérieur. Ces fichiers sont:

- l'entrée standard;
- la sortie standard;
- la sortie d'erreur standard.

Par défaut, ces trois fichiers sont liés au terminal. Un processus lit ses données sur l'entrée standard, écrit ses résultats sur la sortie standard et imprime les erreurs d'exécution sur la sortie d'erreur standard.

I.1.2.5.2. Mécanismes

Un processus est créé par copie du processus créateur. Les deux processus se distinguent par le fait qu'ils n'ont pas le même créateur.

Chaque utilisateur possède un processus shell créé lors du *login*. Ce processus est le père de chaque commande ou programme lancé. Le processus père a la possibilité d'attendre la fin de l'exécution du processus fils ou de le terminer avant.

Le *swapping* est une opération, réalisée par un processus du système, dont le rôle consiste à recopier en mémoire secondaire l'environnement d'un processus ayant perdu le contrôle de l'unité centrale. Le mécanisme de *swapping* permet d'activer de nouveaux processus nécessitant de la ressource mémoire.

En mode noyau, les primitives système employées dans le processus sont exécutées par le noyau du système. La pile noyau du processus et son code exécutable constituent l'espace d'adressage du processus. Exécuter un processus en mode noyau a pour effet de prendre le contrôle de l'unité centrale.

En mode utilisateur, l'espace d'adressage du processus contient: le code exécutable, les données permanentes, la pile des données locales et la zone mémoire allouée dynamiquement lors de l'exécution du processus.

I.1.2.5.3. Communication entre processus

Le système Unix étant multitâche, il peut exécuter plusieurs programmes simultanément. Cette concurrence nécessite souvent l'échange de données entre différents processus. D'autres part, un dispositif de synchronisation entre tous les processus est indispensable afin d'éviter les situations d'interblocage.

Il existe quatre mécanismes traditionnels de communication entre processus: les fichiers, les tubes (*pipes*), les listes FIFO et les événements (signaux):

- **Les fichiers:** les programmes employant des fichiers utilisent une convention pour la synchronisation et pour le format des informations échangées. Habituellement, l'existence ou l'absence de fichier bloqué renseigne sur la disponibilité des données. En ce qui concerne les entrées/sorties avec les fichiers, le système Unix utilise des zones de mémoires tampon; le processus récepteur accède aux données à travers une zone mémoire dédiée à ce genre d'échange. Ce procédé est très efficace pour les systèmes peu chargés. En effet, la quantité d'informations susceptible d'être échangée est égale à la grandeur autorisée des fichiers.

- **Les tubes (*pipes*):** le mécanisme de communication par tubes est le plus classique. La première donnée introduite est toujours la première extraite (FIFO: *First In First Out*). Avec un tube entre deux processus, la sortie standard du premier devient l'entrée standard du second.

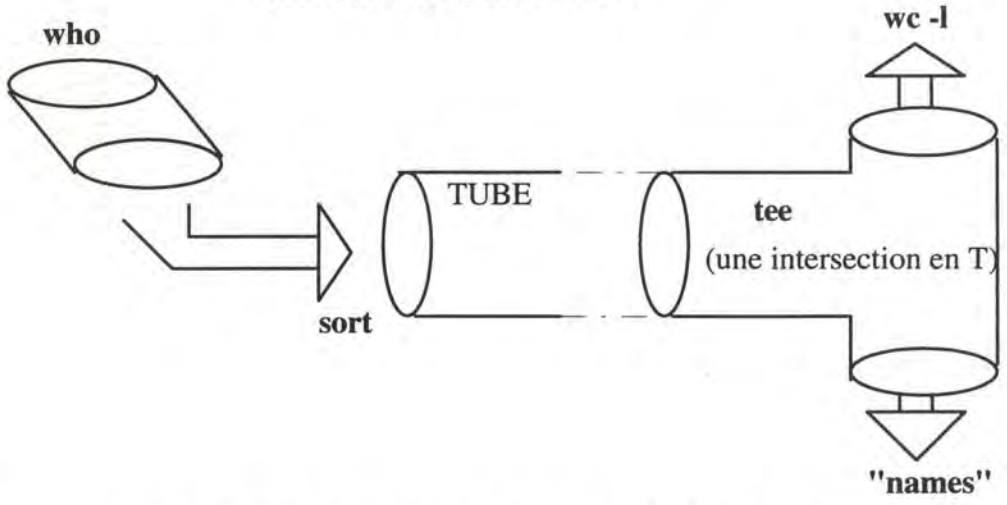
En fait, avec les symboles de redirection ("<" et ">"), on peut rediriger les sorties d'une commande vers un fichier et utiliser le contenu de ce fichier comme entrée pour une autre commande et ainsi de suite. Les fichiers successifs servent uniquement de lieu de passage des données. Ils ne sont plus d'aucune utilité lorsque leurs données ont été envoyées vers la commande suivante, dès lors nous pouvons les supprimer.

Afin d'éviter la création de ces fichiers intermédiaires, nous pouvons utiliser les tubes, qui sont symbolisés par le signe "|". Les données transiteront à travers le tube en sortant d'une commande pour entrer dans la suivante, sans utiliser des fichiers intermédiaires.
L'exemple suivant illustre le fonctionnement des tubes:

Le *"pipelining"* dans Unix:

Pour trouver les noms des utilisateurs courants du système, on entre la ligne suivante:

who | sort | tee names | wc -l



who : génére la liste des utilisateurs connectés au système, le nom des terminaux qu'ils utilisent et la date et l'heure de leur entrée dans le système;

sort : génére (un fichier ou) une liste triée en tenant compte du code asccii;

wc -l : imprime le nombre de lignes (utilisateurs) dans la liste;

"names" : une copie de la liste se trouvera dans un fichier appelé "names".

La commande **tee** sert à créer un tube dédoublé vers deux directions différentes. Ainsi, par exemple le tube **ls -l | tee rep | cat** sera interprété comme suit: le résultat de la commande **ls -l** est envoyé à la fois vers le fichier **rep** et comme entrée à la commande **cat**. Cette dernière commande enverra son résultat vers la sortie standard (l'écran). Avec cette ligne de commandes, le contenu du répertoire de travail sera donc à la fois envoyé dans le fichier **rep** et affiché à l'écran.

Les tubes présentent l'inconvénient d'être connus uniquement par les processus fils; deux processus indépendants ne peuvent communiquer à l'aide de tubes. La synchronisation des échanges par tubes est gérée par le système.

- **Les listes (FIFO):** appelées également "tubes nommés" (*named pipes*), les listes exécutent les mêmes opérations que les tubes ordinaires. Elles possèdent une entrée dans un répertoire et sont utilisables à l'aide d'un chemin d'accès. Ces caractéristiques donnent la possibilité à des processus indépendants de communiquer par ce mécanisme. Pour cela, il faut connaître le nom de la liste et avoir la permission d'accéder à son contenu.

- **Les événements:** Ce mécanisme annonce à un processus l'arrivée d'un événement asynchrone. Dans le domaine des communications, les événements sont employés pour déclencher un processus ou pour synchroniser l'exécution de plusieurs processus. La communication s'effectue à l'aide de la commande *kill*. Le message transmis est identifié par le numéro de l'événement. Cette méthode de communication pose des problèmes quand certains événements sont perdus ou lorsqu'un processus est terminé prématurément. C'est la raison pour laquelle les événements sont employés pour le contrôle de processus et non l'échange de donnée.

I.1.2.6. Caractéristiques d'Unix

Dans toutes ses versions, Unix est devenu un standard pour les systèmes d'exploitations, car il offre une série de caractéristiques qui le rend puissant, souple, polyvalent, et généralement facile à utiliser. Parmi les caractéristiques de Unix on peut citer:

- Le système de fichiers hiérarchique: ce système contient des fichiers et des répertoires. L'arbre du système de fichiers, il est divisé en volumes qu'on peut monter et démonter en les attachant à un des noeuds de l'arbre. Un dispositif de stockage peut contenir un ou plusieurs volumes de système de fichiers.

- L'accès aux fichiers: chaque fichier dans le système de fichiers est protégé par une séquence de bits spécifiée par le propriétaire du fichier. L'accès aux fichiers est alors contrôlé par le système d'exploitation. L'accès à tout le système, sans exception, est accordé à un utilisateur privilégié appelé "super utilisateur" (ou *super user*), habituellement l'administrateur du système.

- Le système de fichiers compatibles, de dispositifs, et d'inter-processus d'entrée/sortie (E/S): Unix s'arrange pour que l'apparence des E/S aux dispositifs physiques et aux fichier soit la même pour tous les programmes utilisateurs. En plus, Unix fournit un mécanisme permettant à l'utilisateur de rediriger les E/S d'un programme, celles-ci peuvent donc être dirigées sur un terminal, un fichier, et même une autre E/S d'un programme. La capacité

de combiner et connecter les programmes existants dans des tubes (*pipelines*) fournit une grande puissance et flexibilité.

- L'efficacité et la concision: le noyau (*kernel)* du système Unix (c'est la partie centrale du système d'exploitation) est relativement petit comparé aux autres systèmes à temps partagés. Autour du noyau central, plusieurs commandes fournissent différentes fonctionnalités. La syntaxe des commandes est généralement concise, ce qui augmente l'apprentissage de l'utilisateur.

- Les processus simultanés: Unix fournit un ensemble de commandes pour amorcer et manipuler les processus asynchrones simultanés. Un utilisateur peut maintenir plusieurs "jobs" à la fois et passer de l'un à l'autre. Les processus simultanés peuvent être connectés par des "*pipes*" pour former un "*pipeline*". De plus, les processus simultanés sont utilisés énormément dans des opérations propres à Unix, ce qui montre l'aspect moderne de ses méthodes.

- Les utilitaires: chaque système Unix apporte plusieurs programmes utilitaires y compris l'éditeur de texte, les compilateurs de langages de programmation, les processeurs de documents, le courrier électronique, la manipulation des fichiers, la base de donnée, le "software engineering", le "networking", etc.

- L'interface utilisateur: le programme de l'interface utilisateur de Unix (le "shell") reçoit les commandes de l'utilisateur et s'occupe de leurs exécutions.

- La portabilité: Unix peut être facilement porté, ou déplacé, à différents ordinateurs. Car Unix est écrit presque entièrement en un langage de haut niveau, le langage C, qui est également portable. Pour porter Unix sur une machine, on a donc besoin du compilateur C.

Le système Unix possède donc plusieurs caractéristiques , les plus importantes sont:

1. capacité multitâches;
2. capacité multi-utilisateurs;
3. portabilité;
4. grande sélection de programmes puissants offerts par le système;
5. communication et courrier électronique;
6. une librairie de programmes d'applications.

Ces caractéristiques sont détaillées ci-dessous:

I.1.2.6.1. Capacité multitâches

C'est le fait d'exécuter plus d'une tâche à la fois. Faire plus d'une chose à la fois sur un ordinateur est aussi du "multitâches". Quand on imprime un fichier et, pendant l'impression, on commence à éditer un autre document, on exécute des opérations multitâches.

Le multitâche nous permet d'exécuter simultanément des tâches, qui auparavant étaient exécutées séquentiellement. L'ensemble des tâches initiales est exécuté plus rapidement, on gagne ainsi du temps.

La grande majorité d'ordinateurs sur lesquels tourne le système Unix ont un seul CPU ou processeur. Ce processeur ne peut exécuter qu'un seul programme à la fois. Ainsi, à un instant donné, le processeur exécute le programme noyau, ou un programme shell, ou un autre programme de commande.

I.1.2.6.2. Capacité multi-utilisateurs

Un système multi-utilisateur permet à plusieurs utilisateurs d'utiliser le même ordinateur simultanément. Plus d'un terminal peut être connecté à un seul ordinateur, et les utilisateurs des terminaux peuvent exécuter des programmes, accéder à des fichiers, et imprimer des documents, tous à la fois.

Le système d'exploitation gère les requêtes faites par plusieurs utilisateurs à l'ordinateur, les sélectionne, les protège des interférences qui peuvent se produire entres eux, et leur affecte des priorités si deux utilisateurs, ou plus, tentent d'utiliser le même fichier de données ou la même imprimante en même temps.

La capacité multi-utilisateur économise du temps en permettant à plusieurs utilisateurs de travailler sur un ensemble d'information simultanément. Ce système multi-utilisateur coûte généralement moins cher que ce que coûte le nombre équivalent de systèmes mono-utilisateur.

I.1.2.6.3. Portabilité du système Unix

Comme on l'a déjà indiqué, le système lui-même est portable car il permet la modification de son code. Et il est plus facile de modifier le code d'un système pour l'installer sur un nouvel ordinateur que d'écrire un nouveau. La capacité de transporter le système UNIX d'une marque d'ordinateur à une autre a été la raison majeure de son acceptation.

Unix tourne sur plus de marques et de types d'ordinateurs, de toutes les tailles, que les autres systèmes. Quand on achète un ordinateur munit du système Unix, on peut ultérieurement le changer avec un autre sans devoir apprendre le nouveau système d'exploitation. Ceci nous fait économiser du temps et de l'argent.

I.1.2.6.4. Les programmes offerts par Unix

Le but de tous les systèmes d'exploitation est presque toujours le même: commander les activités de l'ordinateur. Les systèmes d'exploitation diffèrent dans la façon de faire leur "job" et dans les caractéristiques complémentaires qu'ils offrent.

Unix est unique dans sa conception modulaire, qui permet aux utilisateurs d'ajouter ou supprimer des parties pour répondre à leurs besoins. Les programmes Unix sont comme les pièces d'un "puzzle"; les modules sont fixés ensemble avec des connexions standards. On peut supprimer un module et le remplacer par un autre ou bien étendre le système en ajoutant d'autres modules. Ainsi, chaque système Unix d'un utilisateur est unique.

La majorité des utilisateurs ajoutent ou suppriment des modules suivant leurs besoins, pour renconter leurs besoins. Si on n'a pas besoin d'un module, on pourrait l'écarter sans détériorer le fonctionnement du reste du système. Cette caractéristique est surtout utile dans des implémentations de micro-ordinateur où les capacités des disques sont limitées; la suppression de programmes inutiles laisse plus d'espace pour les fichiers de données.

Unix apporte plusieurs dizaines de programmes. Ces programmes peuvent être divisés en deux classes:

- les utilitaires,
- les outils.

Les utilitaires sont des parties du système Unix qui apportent une assistance nécessaire pour les opérations effectuées par l'ordinateur. Un exemple d'utilitaire est l'interpréteur de commandes Unix, (le "shell"). Sans ce programme les commandes ne peuvent pas être exécutées.

Les outils sont, de leurs cotés, des programmes qui ne sont pas nécessaires aux opérations de base de l'ordinateur, mais offrent une valeur additionnelle à Unix. Ces outils et plusieurs autres programmes d'applications peuvent être achetés séparément. Quelques exemples d'outils: le courrier électronique, les traitements de texte, etc.

I.1.2.6.5. Les communications sur Unix

Il existe plusieurs types de communication sur Unix:

- communication entre plusieurs terminaux attachés au même ordinateur;
- communication entre utilisateurs d'un ordinateur avec ceux d'un autre ordinateur dans le même endroit (ce deuxième ordinateur peut être de marque et de taille différente);
- communication entre plusieurs ordinateurs de tailles et de types différents dans des lieux différents.

La forme la plus simple d'une communication électronique est celle entre des terminaux attachées au même ordinateur. En utilisant le courrier électronique, ils peuvent s'échanger des messages concernant la tenue d'une réunion par exemple ou d'une information quelconque. S'il y avait deux ordinateurs Unix différents, même d'une marque différente, connectés avec des utilitaires Unix, les utilisateurs de ces différents ordinateurs ppourraient envoyer du courrier et échanger des fichiers comme s'ils étaient sur le même ordinateur.

Le système Unix peut également utiliser des lignes téléphoniques pour accomplir des communications entre des ordinateurs situés dans des lieux différents. Il y a plusieurs grands réseaux d'ordinateurs dotés du système Unix à travers les Etats-Unis et l'Europe qui s'échangent des messages et des données.

I.1.2.6.6. Les programmes d'applications des tiers

En plus des programmes d'applications fournis par Bell, plus de 500 programmes d'applications Unix ont été développés et vendus par des tiers (les producteurs d'ordinateurs et de logiciels). Ces programmes ne sont pas une partie du système de base, mais peuvent être achetés séparément et exécutés par le système selon les besoins de l'utilisateur.

Les programmes d'applications Unix, vu leurs aspects multi-utilisateurs et multitâches, peuvent être utilisés par plus d'une personne à la fois. Par exemple: la plupart des programmes de comptabilité nous permettent d'introduire des enregistrements dans les fichiers de paye et d'imprimer simultanément les vérifications.

I.1.2.7. SunOS

I.1.2.7.1. Introduction

Le système d'exploitation (SunOS) des machines Sun a divergé de la version 4.1.BSD du système Unix. C'est avec la version 4.2.BSD que plusieurs services réseaux du SunOS ont été introduits au départ. Heureusement, les concepteurs de Berkeley ont trouvé d'autres alternatives pour lier le tout avec le kernel. Ils ont implémenté les services réseaux en déchargeant certains jobs et en les rendant des daemons spécialisés (des processus serveurs) travaillant en proche coopération avec le kernel. Ainsi, il n'a pas été nécessaire d'ajouter tous les nouveaux codes au kernel lui-même. NFS (pour *Network File System*) est principalement basé sur le kernel (en utilisant seulement un daemon pour faire des appels systèmes), SunOS a continué sur cette ligne de développement. Son domaine d'expansion des services réseaux est uniformément construit sur une architecture qui se base sur un daemon. Exemple de daemons serveurs sont: le portmapper, le NIS (pour *Network Information Service*), le REX (pour *Remote Execution Facility*), le *Network Lock Manager* et le *status monitor*.

I.1.2.7.2. Les services réseau

Pour apprécier la conception des services du système SunOS, il est nécessaire de savoir qu'il est structuré selon le système d'exploitation Unix utilisé sur la plupart des réseaux; il est conçu pour pouvoir évoluer avec les changements des technologies de réseau.

Les services du réseau sont ajoutés au SunOS grâce à des processus serveurs qui sont basés sur le mécanisme RPC (pour *Remote Procedure Call*) de Sun. Ces serveurs sont exécutés sur toutes les machines qui offrent le service. Les daemons Sun diffèrent de ceux qui sont hérités de Berkeley puisque la plupart sont basés sur RPC. Par conséquence, ils bénéficient automatiquement des services fournit par RPC et le XDR (pour *External Data Representation*) qui sont construits dessus.

Tous ce qui est construit avec RPC/XDR est automatiquement une application réseau, comme c'est le cas pour tous ceux qui rangent les données dans les fichiers NFS, même s'ils n'utilisent pas directement RPC. L'environnement RPC/XDR est extensible, en effet de nouveaux services réseaux peuvent être facilement ajoutés à la structure déjà en place. Dans SunOS, les services réseaux sont analogues aux commandes Unix.

I.1.2.7.3. Terminologie

Une machine qui fournit des ressources au réseau est appelée un serveur, cependant, une machine qui emploie ces ressources est appelée un client. Une machine peut être à la fois serveur et client, c'est le cas des ressources NFS (fichiers et répertoires). Une personne qui se connecte sur une machine client est un utilisateur, cependant, un ensemble de programmes qui s'exécutent sur un client sont des applications.

I.1.2.7.4. Les services réseau dominants

1. RPC (*Remote Procedure Call*): cette installation est une librairie de procédures qui offre la possibilité à un processus (le processus appelant) de disposer d'un autre processus (le processus serveur) qui exécute une procédure d'appel, comme si le processus appelant avait exécuté la procédure d'appel dans son propre espace d'adresses. Et comme l'appelant et le serveur sont deviennent deux processus séparés, ils ne doivent pas survivre ensemble sur la même machine physique.

2. XDR (*External Data Representation*): c'est une spécification pour le standard de représentation de données portables. RPC utilise XDR pour assurer que les données sont représentées de la même manière sur les différents ordinateurs, systèmes d'exploitation, et les langages d'ordinateur. Dans SunOS 4.1. XDR est implémenté à travers l'interface sockets, ce qui permet au programmeur d'avoir un accès standardisé aux sockets sans être concerné par des détails de bas niveaux de l'IPC.

3. NFS (*Network File System*): c'est un service indépendant du système d'exploitation qui permet aux utilisateurs de monter des répertoires, même des répertoires root, à travers le réseau pour ensuite traiter ces répertoires comme s'ils étaient locaux. Il y a également une option pour un montage sûr, impliquant l'authentification DES de l'utilisateur et du hôte.

4. *Portmapper*: c'est un service du système auquel sont reliés tous les autres services basés sur RPC. C'est une sorte d'officier de contrôle qui garde des traces de la correspondance entre les Ports (canaux logiques de communication) et les services sur une machine, il fournit une façon standard au client d'améliorer les numéros de Ports de tous les programmes RPC supportés par le serveur. Seulement les programmes RPC qui peuvent l'utiliser.

5. Le service d'information sur le réseau est un service réseau qui a été conçu pour faciliter la tâche de l'administration des grands réseaux. NIS (*Network Information Service*) est un service de base de données distribué, accessible en lecture seulement. Les clients du NFS l'utilisent pour accéder aux données du réseau d'une manière entièrement indépendante des emplacements relatifs du client et du serveur. La base de données NIS fournis typiquement des informations sur les mots de passe, les groupes, les réseaux, et les hôtes.

Sun supporte maintenant les fichiers compatibles du système V et les enregistrements verrouillés tant pour les fichiers locaux que pour ceux montés (NFS).

Les programmes utilisateurs émettent des appels système lockf () et fcntl () pour mettre au point et tester les verrouillages des fichiers - ces appels sont alors traités par les daemons qui

gèrent le verrouillage des réseaux, ceux-ci maintiennent les verrouillages au niveau réseau, même en cas d'échec de plusieurs machines.

Ces daemons sont capables de gérer les échecs des machines car ils sont basés sur un objectif général qui est le *Network Status Monitor*. Ce *Monitor* fournit un mécanisme grâce auquel les applications réseaux peuvent détecter les boutages des machines, dès lors ils peuvent déclencher des mécanismes de redressement spécifiques à l'application. Le gestionnaire de verrouillage est pour cela équipé d'un dispositif de redressement flexible et tolérant.

Il y a d'autres services réseaux qui sont très évidents tels que NIS et REX. Dans ce qui suit nous allons nous concentrer sur le service fondamentale offert par SunOS, le système de fichier du réseau.

NFS (*Network File System*)

C'est un moyen de partager les fichiers dans un environnement de machines hétérogène, dans les systèmes d'exploitation, et dans les réseaux. Le partage est accompli en faisant monter un système de fichiers à distance, ensuite en lisant ou en écrivant dans des fichiers en place.

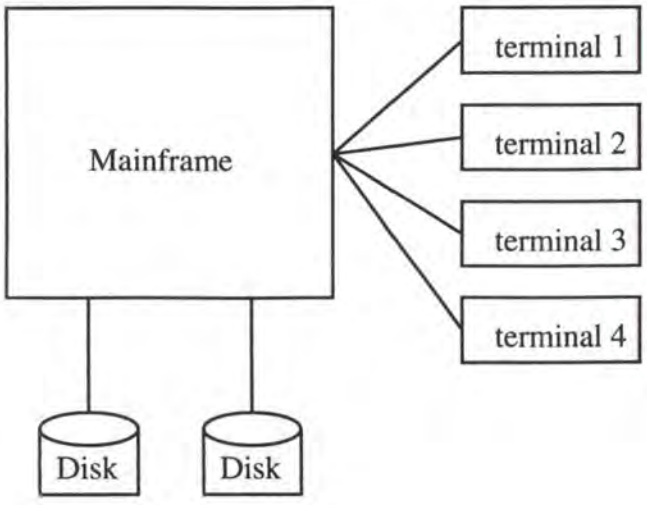
L'indépendance des systèmes d'exploitations ont été considérés comme étant l'objectif de la conception de NFS, tout comme l'indépendance de machines, le redressement des échecs, l'accès transparent et la haute performance.

NFS était ainsi conçu comme étant une collection du services réseau, et non pas comme étant un système d'exploitation distribué. Cependant, il est capable de supporter des applications distribuées sans restreindre le réseau a un seul système d'exploitation.

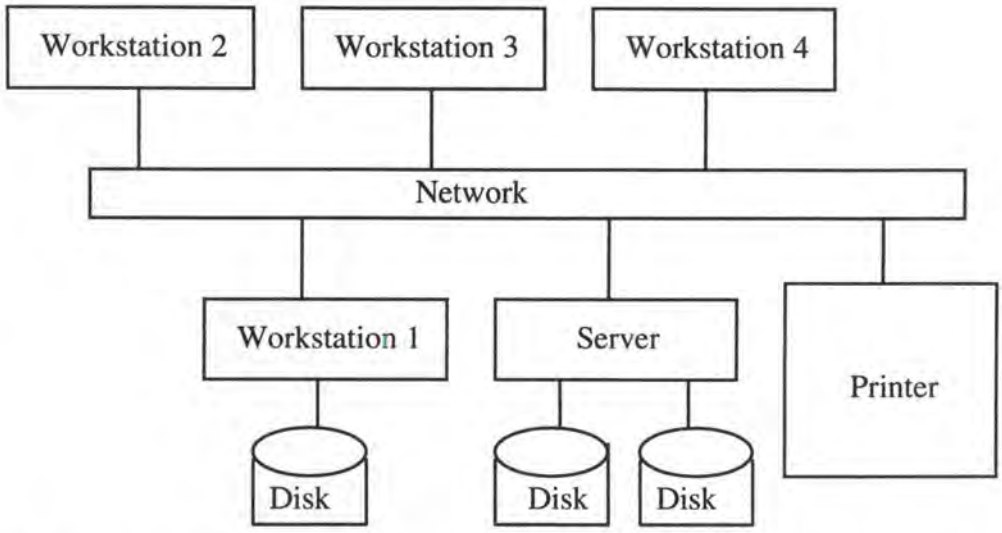
L'implémentation de NFS par Sun est intégrée dans le kernel de SunOS pour des raisons d'efficacité, toutefois une telle intégration n'est pas nécessaire. D'autres vendeurs feront des choix différents, comme dicté par leurs environnements d'exploitation et applications. Donc grâce à la conception ouverte de NFS toutes ces applications seront capables de tourner ensemble sur un seul réseau.

⇨ **Les environnements informatiques**

L'environnement en temps partagé traditionnel se présente comme suit:



Le problème majeur avec cet environnement est la compétition pour les cycles CPU. L'environnement de la station de travail résout ce problème, mais exige plusieurs lecteurs disques. Un environnement de réseau est illustré dans le schéma ci-dessous:



L'objectif de la conception de NFS était de rendre tous les disques disponibles. Les stations de travail individuelles ont accès à toutes les informations se trouvant sur le réseau. Les imprimantes ainsi que les super-ordinateurs pourraient également être disponibles sur le réseau.

⇨ L'architecture d'NFS

1. L'accès transparent à l'information: les utilisateurs sont capables d'accéder directement aux fichiers qu'ils désirent consulter sans connaître l'adresse réseau de cette donnée. Pour l'utilisateur tous les systèmes de fichiers montés avec NFS lui semblent comme des disques privés. Il n'y a pas de différences apparentes entre la lecture ou l'écriture sur un disque dans un endroit ou un autre. L'information dans le réseau est bien distribuée.
2. Les différentes machines et les systèmes d'exploitation: des services appropriés doivent être intégrés dans un réseau car aucun vendeur ne peut fournir des outils valables pour tous les travaux demandés. NFS fournis, pour cette intégration, une plate-forme qui est flexible et indépendante du système d'exploitation pour réaliser cette intégration.
3. La facilité d'extension: un système distribué doit avoir une architecture qui permet l'intégration de nouvelles technologies sans perturber l'environnement déjà existant. L'approche des services réseau de NFS ne dépend pas du système d'exploitation utilisé dans le réseau, elle offre un ensemble extensible de protocoles d'échanges de données, elle supporte l'intégration flexible des nouveaux software.
4. La facilité d'administration du réseau: l'administration des grands réseaux peut être très compliquée et peut consommer beaucoup de temps. Idéalement, leur administration devrait être, du moins pour l'administrateur, aussi simple qu'un ensemble de systèmes de fichiers locaux sur un système à temps partagé. Le système Unix possède un ensemble de commandes de maintenance, un service d'information sur le réseau et un service de base de données réseau basés sur NFS. Ces services sont adaptés et élargis pour des objectifs d'administration de réseau de machines. Le NIS (*Network Information Service*) permet également certains aspects de l'administration de réseau qui peuvent être centralisés sur un

petit nombre de serveurs de fichiers. L'interface NIS est implémentée en utilisant RPC et XDR, elle est donc disponible sur des systèmes d'exploitation différents de Unix et pour des machines différentes de Sun. Les serveurs NIS n'interprètent pas les données, il est donc facile aux nouvelles bases de données d'être ajoutées au service NIS sans modifier les serveurs.

5. La fiabilité: la fiabilité d'NFS provient de la robustesse du système de fichier 4.2.BSD, des protocoles NFS, et de la méthodologie basée sur le daemon par laquelle les services réseaux telles que le blocage de l'enregistrement et de fichiers, sont fournis. L'avantage principal d'un serveur, dont chaque transaction traitée est indépendante, est la robustesse vis-à-vis du client, du serveur, ou des échecs du réseau. Cette robustesse est surtout importante dans un réseau complexe composé de systèmes hétérogènes, dans lequel plusieurs d'entre eux ne sont pas sous le contrôle de l'administrateur, et qui peuvent tourner sur des systèmes non testés.
6. La haute performance: la flexibilité d'NFS permet une configuration de échanges de coût et de performance. Par exemple, la configuration des serveurs avec des grands disques performants, et les clients sans disques, pourrait rapporter une meilleure performance à coût moins élevé que d'avoir plusieurs machines ayant des petits disques bon marché. De plus, il est possible de distribuer les données du système de fichier sur plusieurs serveurs et bénéficier du multitraitement sans perdre la transparence. Dans le cas des fichiers ouverts en lecture seulement, les copies peuvent être gardés sur plusieurs serveurs pour éviter les bouchons. Sun a également rajouté plusieurs améliorations de performance à NFS, telles que "*fast path*" pour les opérations clés, les services asynchrones des requêtes multiples, le cache de verrouillage du disque, et la lecture en avant et l'écriture en arrière asynchrones dans le but de réduire l'impact d'un échec non anticipé.

⇨ L'implémentation de Sun NFS

Dans l'implémentation du Sun NFS il y a trois entités à prendre en considération: l'interface du système d'exploitation, l'interface du système de fichier virtuel (VFS), et l'interface du NFS. L'interface du système d'exploitation Unix a été préservé dans l'implémentation d'NFS par Sun, pour assurer la compatibilité des applications existantes. Les applications utiliseront *read()* et *write()* pour accéder aux fichiers NFS comme pour accéder aux fichiers locaux.

Le VFS est vu comme une couche que Sun a ajouté autour du système de fichier Unix traditionnel. Ce système de fichiers traditionnels est composé de répertoires et de fichiers, chacun possède un *inode* (index node), contenant des informations administratives du fichier, telles que son emplacement, sa taille, son propriétaire, les permissions, et le temps d'accès. Des nombres sont assignés aux *inodes* pour les identifier dans le système de fichier, mais un fichier dans un système de fichiers peut avoir le même nombre qu'un autre fichier sur un autre système de fichiers. Ce qui est un problème dans l'environnement du réseau, car les systèmes de fichiers distants doivent être montés dynamiquement, et le numérotage des conflits pourrait causer des ravages. Pour résoudre ce problème, Sun a conçu le VFS, qui est basé sur une structure de données appelé *vnode*. Dans le VFS, les fichiers sont désignés par des numéros uniques, même dans le réseau. Les *Vnodes* séparent proprement les opérations du système de fichier de la sémantique de leur implémentation. Au-dessus de l'interface, le système d'exploitation traite avec des *vnodes*; au-dessous de cet interface, le système de fichier peut implémenter les *inodes*.

L'interface VFS peut connecter le système d'exploitation à une variété de systèmes de fichiers (par exemple, 4.2BSD ou MS-DOS).

Le VFS définit et implémente l'interface NFS sur la base des mécanismes RPC et XDR. Dans le cas d'accès au VFS local, les requêtes sont adressées aux systèmes de fichiers de données sur des dispositifs connectés aux machines du client. Dans le cas d'accès aux VFS distants, la requête est adressée aux couches RPC et XDR sur le réseau. Dans l'implémentation courante, Sun utilise les protocoles UDP/IP (cf. partie II) et le support Ethernet (cf. I.1.1.). Du coté du serveur, les requêtes seront passées dans les couches RPC et XDR jusqu'à leur arrivée au serveur NFS; le serveur utilise les *vnodes* pour accéder à un de ses VFS locaux et ainsi traiter la requête.

L'implémentation de NFS par Sun fournit cinq types de transparences:

1. Le type du système de fichiers: Le *vnode*, en conjonction avec un ou plusieurs VFS locaux permettent à un système d'exploitation (client et application) d'avoir une interface transparente avec une variété de types de systèmes de fichiers.
2. L'emplacement du système de fichiers: comme il n'y a pas de différenciation entre un VFS local et un VFS distant, l'emplacement des données du système de fichier est transparent.
3. Le type du système d'exploitation: le mécanisme RPC permet l'interconnexion d'une variété de systèmes d'exploitation sur le réseau, et rend transparent le type du système d'un serveur distant.
4. Le type de machine: la facilité de définition du XDR permet la communication d'une variété de machines sur le réseau et rend transparent le type de machine du serveur distant.
5. Le type de réseau: RPC et XDR peuvent être implémentés pour une variété de protocoles de transport, rendant ainsi transparent le type de réseau.

Des implémentations d'NFS plus simple sont possibles aux dépends de quelques avantages de la version Sun. En particulier, un client (ou un serveur) peut être ajouté au réseau en implémentant un seul coté de l'interface NFS. Un avantage de l'implémentation Sun est que le client et le serveur peuvent être symétriques; ainsi, il est possible pour chaque machine d'être un client, un serveur ou les deux à la fois. Les utilisateurs des machines clientes avec disques peuvent s'arranger pour les partager sur NFS sans faire appel à l'administrateur du système ou sans configurer un système différent sur leurs stations.

Chapitre I.2.

LA SECURITE

ET SES OUTILS

I.2.1. Introduction: le problème de la sécurité en général

I.2.1.1. La nécessité de la protection

La quantité d'informations traitées actuellement grâce à un ordinateur a plus que décuplé depuis quelques années. Il suffit de regarder la capacité des disques durs équipant, par exemple, les PCs. De plus, dans le cas d'un réseau, l'ensemble des informations circulant est en général stocké en un seul point: le(s) disque(s) dur(s) du serveur. Dès lors, il est évident que si des précautions n'étaient prises, un problème, le plus petit soit-il, sur le serveur pourrait paralyser l'ensemble des utilisateurs.

I.2.1.2. La qualité souhaitée

Les systèmes actuels sont loin d'être totalement sécurisés puisqu'ils sont de plus en plus complexes, automatisés, distribués et intégrés entre eux. Leurs sécurités posent un problème assez difficile puisqu'aucun système de sécurité ne garantit une protection totale. Malgré la diminution de certains risques, plusieurs autres subsistent.

L'objectif de base de la sécurité des systèmes informatiques est de maintenir trois qualités fondamentales:

1. la confidentialité: qui consiste à protéger l'information et la garder secrète;
2. l'intégrité: qui consiste à conserver l'information et la garder intacte, ainsi elle ne subit de modifications que par l'intervention des utilisateurs autorisés;
3. la disponibilité: qui consiste au bon fonctionnement du système, ainsi les ressources et les informations doivent être accessibles dans des délais convenables par les utilisateurs autorisés.

I.2.1.3. Les différents types de dangers

⇨ **sécurité physique et sécurité logique:** La sécurité informatique se divise en deux catégories: la sécurité physique (conditionnement d'air, incendie etc.) et la sécurité logique (contrôle d'accès, protection de l'information etc.). On ne tiendra compte dans ce mémoire que de la sécurité logique.

Dans cette section on traitera des différents types d'attaques et leurs contre-mesures. L'analyse des fichiers de traces générés par le système, objet de ce mémoire, est une de ces contre-mesures.

⇨ **détérioration des données:** Dans un réseau, malheureusement, des détériorations de données peuvent provenir de bien d'autres endroits, comme nous allons le montrer tout au long de ce chapitre.

- **Nombre d'utilisateurs:** Bien que cela ne soit pas évident au premier abord, il n'est pas toujours raisonnable d'installer un poste de travail sur le réseau pour chaque personne devant travailler avec un ordinateur. En principe pourtant, si le système de protection des programmes et de gestion d'accès au réseau est bien conçu, cela ne devrait pas poser de problèmes. Néanmoins lorsque le logiciel n'est pas assez performant, il peut s'avérer nécessaire d'empêcher l'emploi d'une station de travail à certaines personnes. En effet, lorsqu'il s'agit de réseau, chaque utilisateur a une certaine responsabilité envers les autres; en ceci qu'une erreur de la part d'une seule personne peut paralyser tout le monde. Dès lors, quelqu'un ne maîtrisant pas bien les bases de l'informatique devrait, idéalement, faire ses premières armes sur un PC isolé ou un poste de travail mais, alors, suivi en permanence par un habitué.

- **Topologies et méthode d'accès:** Certaines méthodes d'accès offrent un niveau de sécurité plus élevé que d'autres. Une méthode où les collisions sont quasiment impossibles est, bien sûr, préférable, mais une méthode qui les prévoit, les détecte et les rend inoffensives est probablement ce qu'il y a de mieux. Néanmoins, ce choix, celui de la CSMA/CD (cf. I.1.1.) n'est pas toujours possible pour des raisons de dispersion des stations de travail, de logiciels imposés, etc. Dès lors, une attention toute particulière doit être apportée lors du choix d'une topologie et donc de la méthode d'accès. N'oublions pas que topologie et méthode d'accès sont intimement liées au type de traitement des erreurs de transmission et collision pouvant se produire sur un réseau informatique.

- **Protection software:** Il s'agit ici de la méthode la plus efficace de protection de données. La protection logiciel se passe à différents niveaux. La première précaution à prendre est d'exiger un nom utilisateur sur chaque station de travail. Lorsqu'il voudra se connecter au réseau, l'utilisateur devra ainsi s'identifier en donnant par exemple son nom, un numéro ou tout autre code, le plus simple restant le nom. Cela peut paraître assez naïf de donner le nom comme identification, mais étant donné qu'en général un mot de passe est demandé par la suite, cela permet plus facilement de savoir qui s'est connecté sur le réseau et éventuellement à quelle heure. Il est généralement prévu dans le logiciel gérant le réseau de garder trace de ces noms. Par la suite, on pourra ainsi déterminer un taux d'utilisation du réseau pour chacun afin de mettre en évidence les heures d'affluence. On pourra aussi consulter ce "calendrier" d'accès au réseau pour vérifier la présence de personnel, les accès éventuels en dehors des heures de travail etc.

De plus, dans un souci purement économique, ce calendrier de connexion machine peut permettre de déterminer les redevances à réclamer dans le cadre de location de certaines machines se branchant sur le réseau.

Après cette première étape, un mot de passe secret, cette fois, doit permettre à chaque utilisateur d'être sûr que personne ne viendra se connecter en son nom.

La combinaison des deux bonnes "réponses" - nom d'utilisateur et mot de passe - donne alors enfin accès au réseau. Si l'une des réponses est fausse, l'accès est tout simplement refusé, des essais successifs étant possibles. Généralement, après quelques tentatives l'accès est définitivement interdit.

I.2.2. Le problème de la détérioration des données

I.2.2.1. Le problème

L'accès au réseau ne signifie pas disposer d'une liberté totale. En effet, on peut, sur la plupart des gestionnaires de réseaux, déterminer un certain nombre de droits aux utilisateurs. Certains pouvant avoir tous les droits (lecture, écriture, effacement...), d'autres seulement des droits limités (cf. I.1.2.). Ces droits portent aussi sur l'accès aux répertoires du disque dur.

I.2.2.2. Quelques approches de protection

A un autre niveau plus avancé, le gestionnaire de réseau peut prévoir des systèmes de sécurité des données assez évolués tels que, par exemple, le Netware de Novell (que l'on appelle un *Track Transaction System* ou TTS).

Plus évoluées encore: les fonctions de *mirroring-duplexing*. Dans le cas du *mirroring*, il s'agit en fait du dédoublement de la partie vitale d'un serveur de fichier: le disque dur. Les données devant être stockées sont envoyées sur un contrôleur particulier de disque dur et, via celui-ci, sur deux disques durs différents en parallèle. Chaque support magnétique étant alors une copie conforme à l'autre. Si un problème devait survenir sur l'un des disques durs, le programme le signalerait et proposerait à l'utilisateur de continuer ou non. Le *duplexing* est une version plus poussée du *mirroring*. Ici le serveur est équipé de deux contrôleurs et de deux disques durs.

Poussant toujours plus loin, les concepteurs de gestionnaires de réseau ont commercialisé un *operating system* de réseau où le serveur de fichier est tout simplement en double!!! (*server mirroring*). Plus d'inquiétude en cas de défaillance d'un des serveurs.

I.2.2.3. Une classification des niveaux de protection

En fonction du type de logiciel choisi, il est proposé différentes solutions de protection, qui deviennent de plus en plus complexes et nombreuses quand on s'élève dans le haut de gamme. Ces logiciels de protection sont souvent classifiés. Il existe trois niveaux (*level*) de classification. Le premier niveau regroupe les logiciels qui permettent le dédoublement de la FAT (*File Attribute Table*), la recherche rapide dans la FAT, la gestion des mauvais secteurs et la recherche intelligente des données. Le deuxième niveau regroupe ceux possédant la protection comprise dans le niveau 1 et y ajoute des disques durs en miroir, des disques durs et des contrôleurs en dédoublement et le TTS (*Transaction Track System*). Le troisième niveau regoupe les logiciels qui offrent tout ce qui précéde et ajoutent le dédoublement de serveur, le câble et les systèmes auxiliaires de sécurité.

Les systèmes de protection software les plus rencontrés sont, par exemple, ELS 1, ELS 2 et AN 286 de Novell. Ces softwares sont repris sous le niveau de protection 2; le logiciel 386 Vs

3.1 est, quant à lui, sous le niveau de protection 3. Les trois niveaux de protection sont décrits ci-dessous:

❑ **Level 1:** Cache mémoire du disque (*Disk caching*); le système recopie la FAT (*File Attribute Table*) - sorte de plan des fichiers sur le disque dur - du disque dur en RAM; ce qui permet lors de la recherche d'un fichier de lire la FAT non pas sur le disque dur, mais en RAM. On obtient de la sorte un gain de temps d'un facteur 100. Ainsi, plus on travaille avec un disque dur offrant une grande capacité, plus sa FAT sera importante et plus il sera nécessaire d'augmenter la capacité RAM.

1. Dédoublement de la FAT (*Dual FAT*): Le système recopie la FAT sur le disque dur en deux endroit: une FAT en bord de disque (piste zéro), l'autre FAT au centre du disque (piste max.), et ce pour deux raisons:

- **sécurité:** si une FAT est détruite, on lit l'autre;
- **rapidité:** car il y a une gestion dynamique et intelligente des têtes du disque dur et ces dernières accèdent à la FAT la plus proche. Les FATs sont toujours l'image l'une de l'autre par un système de sauvegarde permanente.

2. Recherche rapide dans la FAT (*Hashing*): Un algorithme, géré par le système, recherche les noms des fichiers dans FAT suivant une table numérique créée par lui. Ce qui permet un accès direct au lieu d'un accès séquentiel de la FAT; le gain de temps est estimé à 40%.

3. Gestion des mauvais secteurs (*Hot Fix*): Le système gère les mauvais secteurs du disque dur. Lorsqu'un fichier doit être écrit sur le disque dur, il est relit directement après l'inscription. La lecture est comparée avec le fichier original (*read after write*). Si une erreur intervient, le secteur où apparaît l'erreur est déclaré mauvais. L'adresse de ce secteur est gérée dans une table créée à cet effet qui va recopier les données qui auraient dû être contenues dans le mauvais secteur sur un autre endroit du disque dur (*Redirection Data*); et ceci en maintenant en permanence la corrélation entre les adresses de redirection.

4. Recherche intelligente de données (*Elevator Seeking* ou *Skew Factor*): Le système gère les secteurs d'un même fichier sur des pistes différentes, de telle sorte que lors de la lecture ou de l'écriture, l'ordre soit séquentiel et se fasse dans le sens de rotation du disque. Ceci, une fois encore, dans un souci de gain de temps.

❑ **Level 2:** Ce niveau de protection reprend tous ceux compris dans le niveau 1 auxquels on ajoute:

1. Disque dur en Miroir (*Mirroring*): Pour être certain de posséder un backup des données contenues sur le disque dur, on installe un deuxième disque dur qui est le reflet permanent du premier.

2. Disque dur et contrôleur en dédoublement (*duplexing*): Pour bénéficier des mêmes avantages que le mirroring et pour prévenir tout problème en cas de défectuosité du contrôleur de disque dur, on dédouble le disque dur ainsi que son contrôleur.

3. TTS (*Translation Track System*): Cette protection assure le lien permanent entre, par exemple, les fichiers et les fichiers d'index d'une base de données. En effet, un fichier d'index dispose d'un élément de chaînage avec les fichiers principaux. Lorsqu'un élément nouveau est apporté dans la base de donnée et que l'on réindexe les fichiers, il y a une réorganisation totale

des éléments à indexer avec, comme seul lien avec les données, l'élément de chaînage. Si, à cet instant, une interruption intervient, le chaînage risque d'être perturbé et l'indexage détruit.

Grâce au TTS une table est créée, elle reprend toutes les données et les liens avec l'index. De sorte que si un incident pareil devait arriver, seul l'encodage des dernières données serait perdu car les index précédant la dernière modification seront toujours cohérents.

❑ **Level 3:** Ce dernier stade de la protection reprend tous les précédents auxquels vient s'ajouter:

Dédoublement de serveur (*server duplexing*): Ceci est le top niveau de sûreté, car il consiste à installer un deuxième serveur exactement à l'image du premier. De plus, pour augmenter encore la sécurité, il est possible de brancher un UPS (cf. I.2.1.2.) qui assurera l'alimentation du serveur. C'est en fonction des applications, des besoins, des nécessités et du budget que l'installateur choisira son degré de protection qui, en général, est d'autant plus poussé que la version du logiciel est sophistiquée.

Le câble: L'information sur le réseau est transportée sur du câble (cf. I.1.1.). Généralement, trop peu d'attention est accordée dans le choix de tel ou tel câble. Le choix du type de câblage n'est pas l'objet de ce paragraphe, mais bien le choix de la qualité à l'intérieur d'un type de câble. Les différences de prix entre un média bien isolé et sommairement protégé ne sont que très faibles, et ceci certainement dans le cas de petites et moyennes entreprises, où les distances à câbler ne sont pas très grandes. Dès lors, ce support de l'information en transit, qui reste par ailleurs installé dans des endroits pas toujours chauffés ou exposés à des chaleurs fortes ou à l'humidité, mérite bien qu'on lui prête un peu d'attention. Chaque constructeur fournit généralement des données concernant le type d'isolation du média et les résistances de celui-ci ainsi que les conditions extrêmes de bon fonctionnement de son produit.

A un autre échelon, hormis ces protections, il faut également prendre en considération la protection contre l'espionnage (lecture illégale des données circulant sur le câble) industriel ou militaire par des espions-pirates. Une des méthodes pour pallier ce problème, est l'emploi d'un algorithme d'encryptage des données connu seulement par l'émetteur et le récepteur du message.

Les systèmes auxiliaires de sécurité: On n'est jamais trop prudent. Une copie des fichiers les plus importants sur disquettes est une pratique qui devrait maintenant être un réflexe chez l'utilisateur consciencieux. Les disques durs des réseaux pouvant atteindre de grandes capacités, il n'est pas possible de faire une copie de sécurité (*backup*) de tous les fichiers, qu'ils soient de données ou de programmes, sur des disquettes.

Dans certains cas, il faudrait plus d'une centaine de disquettes et beaucoup de patience. Dès lors, une copie de sécurité sur bande magnétique (*tape streamer*) devrait être envisagée. Ces systèmes de sauvegarde permettent, en effet, de transposer un grand nombre de données en un temps minimum et avec une facilité qui ne cesse de croître. Si des disques durs locaux représentent une grande importance, il n'est pas nécessaire de disposer de plusieurs systèmes magnétiques, car ceux-ci sont assez petits et donc facilement transportables d'une station à l'autre. Enfin, est-il besoin de le rappeler ? une alimentation de secours avec autonomie nous semble être un must pour les entreprises désireuses de se mettre à l'abri d'erreurs provenant de parasites, de différences de tension sur le réseau ou absence subite d'alimentation.

I.2.3. Le problème de l'alimentation

Depuis longtemps, des interfaces d'alimentation équipent les gros systèmes informatiques, centres nerveux de grosses entreprises. Ces interfaces d'alimentation sont souvent plus connues sous le nom générique d'UPS, (pour *Uninterrupted Power Supply*) ou alimentation continue. Afin de déterminer le type d'alimentation dont l'utilisateur a besoin, une petite analyse très rapide du problème est nécessaire. La première étape consiste à trouver l'origine et le type de perturbation.

⇨ **Les perturbations:** Elles peuvent être de gravité et d'origine très différentes. D'une manière générale, on pourrait les classer en deux types:

- les coupures apparentes;
- les perturbations "invisibles".

Les premières sont assez évidentes. Concrètement, alors que vous êtes en train de travailler, votre ordinateur s'arrêtera subitement en plein milieu de l'application pour recommencer à travailler au point de départ, c'est à dire comme si vous veniez d'allumer votre ordinateur. On peut facilement imaginer les dégâts qu'une telle coupure peut provoquer dans les données. Certains programmes permettent cependant d'effectuer des copies de sécurité régulières et automatiques en fonction d'un facteur temps que l'on doit indiquer. La durée de la coupure peut être assez long. Une telle coupure peut durer plusieurs minutes et mêmes plusieurs heures pendant lesquelles l'utilisation des PCs, par exemple, est impossible. De plus, quelque soit la durée de la coupure, cette dernière anéantit tous travaux non sauvegardés et il y a risque de destruction de certains fichiers "ouverts" à l'instant de la coupure.

Les perturbations les plus dangereuses sont néanmoins les moins perceptibles. Si l'effet n'est pas directement visible, il peut devenir manifeste beaucoup plus tard, ce qui a pour conséquence que l'utilisateur ne fait pas le rapprochement entre le problème et la perturbation électrique. Dès lors, il va continuer à travailler comme si ne rien était, n'étant pas conscient du problème. Les erreurs qui apparaissent ultérieurement, dans un programme par exemple, peuvent aboutir à des résultats aberrants. Lorsque l'erreur est enfin visible, il faut encore en trouver l'origine: la cause de l'erreur peut aussi bien être une donnée "faussée" qu'un bit changé dans le programme lui-même.

⇨ **Les causes:** Elles peuvent être de nature assez diverses. Les causes des coupures et perturbations les plus apparentes sont aussi souvent les plus faciles à identifier. S'il y a coupure, c'est l'installation locale de l'utilisateur qui doit être la première mise en cause dans bien des cas, quoique des orages, par exemple, puissent aussi parfois provoquer des perturbations. Les causes des perturbations non directement perceptibles sont plus diverses. Il peut s'agir d'une coupure générale rare et souvent très courte que l'on appelle micro-coupure. En campagne, une usine est une cause d'ennuis à ne pas sous-estimer. En effet, étant donné la capacité de "pompage" d'une telle usine sur le réseau électrique, les installations proches n'ont plus la même qualité ou quantité de courant. Celui-ci peut être diminué en amplitude ou augmenté brusquement et démesurément, dépassant ainsi la tolérance de variation au niveau de l'alimentation de l'ordinateur. D'autre part, la qualité de la tension peut aussi être diminuée.

En ville, on pourrait avoir tendance à se sentir protégé de ces perturbations. Ce qui est une illusion. De tels problèmes peuvent survenir à proximité d'une administration, par exemple, où en quelques minutes plusieurs dizaines ou centaines de machines (ordinateurs, photocopieurs, etc.) vont être simultanément mises en service.

⇨ **Les conséquences:** Au niveau de la qualité et de l'amplitude de la tension les conséquences sont assez simples: diminution de la qualité par la présence de parasites et variations extrêmes au niveau de l'amplitude. Dans un ordinateur, les conséquences peuvent apparaître directement, mais aussi indirectement, à plus long terme. Si une coupure brusque se produit, c'est tout l'équipement électrique et électronique qui sera paralysé, entraînant ainsi une perte de rentabilité de l'entreprise, et ceci sans compter le temps nécessaire pour récupérer les fichiers endommagés (si cela est encore possible). Des pertes de fichiers ou erreurs dans le traitement des données peuvent se produire. Il suffit d'imaginer une coupure lors de l'indexation d'un fichier ou du calcul de balance dans un programme de comptabilité. Pire encore, les parasites et variations d'amplitude peuvent produire des dégâts sur les composants électroniques déjà sensibles.

Un ou plusieurs bits modifiés peuvent complètement altérer le déroulement d'un programme et en fausser les résultats

⇨ **Les solutions:** Une première solution est d'envisager une protection sans autonomie. La plus simple se présente sous forme de blocs multi-prise contenant un filtre. Plus évolué, le conditionneur de réseau (le réseau électrique et non le réseau informatique) remplit les fonctions d'isolation contre les perturbations du milieu (moteurs, néons, etc.), de régulation de tension ainsi que la fonction de filtrage. Plus coûteux et plus efficaces, les systèmes de protection avec autonomie peuvent être classés en deux catégories.
Les IPS ou *Interrupted Power Supply*, ou encore interfaces d'alimentation interrompue souvent abusivement reprises sous le terme générique d'UPS (*Uninterrupted Power Supply*).
Le principe des IPS est simple. L'ordinateur est branché sur l'IPS qui est lui-même branché sur le réseau électrique. La tension entre dans l'IPS et l'ordinateur est filtrée. Parallèlement, la batterie est maintenue chargée. Lorsque la qualité de la tension n'est plus acceptable ou lorsqu'il y a coupure, l'IPS prend le relais et donne une autonomie qui peut aller de plusieurs minutes à plusieurs dizaines de minutes selon les types. Lorsque des conditions acceptables de courant sont à nouveau présentes, la batterie de l'IPS se déconnecte pour être rechargée, l'interface d'alimentation ne remplissant alors plus qu'une fonction de filtrage des parasites. On le voit, l'IPS est un système de protection avec autonomie et interruption. Il y a en effet un délai de l'ordre de quelques millisecondes entre le moment où un manquement est détecté et le moment où la batterie est mise en fonction. Ce délai ne devrait jamais dépasser les dix millisecondes, durée maximale pendant laquelle des composants peuvent ne plus être alimentés. Généralement, les constructeurs annoncent des interruptions de deux à quatre millisecondes lorsqu'on parle d'IPS.

Les UPS fonctionnent de manière à peu près similaires. La différence principale, et d'importance, est qu'il n'y a pas d'interruption entre le moment où il y a un manquement et le moment où la batterie prend le relais. D'autre part, dans les UPS, le courant est régénéré pour lui donner toutes les qualités qu'il avait au départ de la centrale électrique, alors que l'IPS n'effectue qu'un filtrage. Une première chose à faire après avoir décidé de l'achat d'un système de protection, est d'évaluer sa puissance. Dans le cas d'un réseau, le premier poste auquel on doit penser est, bien sûr, le serveur afin de pouvoir sauvegarder les fichiers. Si certaines

stations réalisent un travail important telle que comptabilité ou mise à jour de gros fichiers, il pourrait être prudent de le protéger.

Le choix entre un UPS et un IPS est très délicat, l'idéale étant bien évidemment d'équiper un maximum de postes et le serveur. De préférence, le serveur sera protégé par un UPS alors que les stations moins importantes le sauront par des IPS. Ce choix est discutable et il s'agit ici plutôt d'une option prudente.

L'utilisation d'une interface d'alimentation est assez simple. Les cartes que l'on installe dans le PC permettent, en cas de passage sur l'interface d'alimentation, de le signaler à l'utilisateur par un message à l'écran. Ces systèmes d'UPS *Monitoring* réalisent une séquence de fermeture des fichiers ouverts et permettent ainsi de les sauvegarder et ce automatiquement, sans intervention humaine.

Le but d'une protection contre une coupure de courant ou perturbation du réseau électrique n'est pas de continuer à travailler sans interruption, mais bien de pouvoir arrêter proprement et dans les quelques minutes qui suivent, selon la capacité d'autonomie choisie. Aussi, une interface d'alimentation protège continuellement contre les variations de tension. Lorsque celles-ci deviennent trop importantes, la protection se met en marche.

Pour conclure, nous dirons qu'il nous semble que pour avoir une bonne sécurité du matériel, un réseau devrait être équipé d'une protection contre les coupures et variations du réseau électrique au moins au niveau du serveur. Une protection plus forte serait efficace mais chère, une protection trop faible serait purement et simplement inefficace.

I.2.4. Les attaques malicieuses

I.2.4.1. Les différents types d'attaques

Plusieurs types d'attaques peuvent menacer les systèmes informatiques. Dans la littérature, plusieurs auteurs, tel que Lunt, ont classifié ces formes d'attaques dans le but de définir des catégories de scénarios. Trois classes déterminent les grands types de menaces [Asax 93a]:

⇨**Les intrusions externes:** ce sont les intrusions effectuées par des utilisateurs non autorisés à utiliser le système (Ex. tentative d'intrusion à l'aide d'un système générateur de mots de passe). Un symptôme caractéristique de ce type d'intrusion est par exemple l'échec répété de la commande login.

⇨**Les intrusions internes:** ce sont des intrusions effectuées par des utilisateurs autorisés à utiliser le système. Parmi celles-ci on peut distinguer:

- Ceux qui cherchent à outrepasser leurs privilèges afin d'atteindre des ressources ne leur étant pas accordées (Ex. consultation de fichiers protégés, piratage de programmes etc.). Le symptôme dans ce cas pourrait être l'échec répété des commandes d'accès ou de changement de mode d'accès à des fichiers appartenant à d'autres utilisateurs;

- Ceux qui travaillent sous une fausse identité. Et ceci en masquant leur identité pour ne pas être détecté lors d'un acte malveillant. La détection de ce type d'intrusion nécessité

une description adéquate des activités normales des utilisateurs (leurs profiles). Ainsi, la déviation de ce profile peut être considéré comme un symptôme d'intrusion;

• Ceux qui abusent de leurs pouvoirs et font de leurs droits d'accès un usage excessif. De telles violations de la sécurité sont difficilement détectables dans l'analyse des fichiers de traces.

⇨**Les virus informatiques:** les virus sont caractérisés par une succession d'événements (un scénario). (Ex. l'existence de quelques séquences d'instruction destinés à détruire et/ou altérer les données). Pour détecter des virus inconnus on doit disposer d'une description large et détaillée des activités normales du système, ce qu'on appelle le profile du système. Ce profile décrit les moyens du CPU, le nombre moyen des opérations de lecture/écriture de chaque fichier exécutable. Plusieurs formes de virus peuvent être rencontré: les virus logiques, les chevaux de Troie, les bombes logiques, les vers et les bactéries. Une description sommaire de ces virus est donnée ci-dessous:

• Les virus logiques se présentent comme une partie de code conçue dans le but de détruire un logiciel cible tout en s'auto-copiant. Ce mode de reproduction permet au virus d'infecter ainsi tous les types de supports, disques, disquettes, réseaux, mémoires etc.

• Les chevaux de Troie sont des programmes conçus dans le but de voler des informations relatives à la sécurité d'un réseau (modification, destruction de documents etc.). Les chevaux de Troie ne se reproduisent pas.

• Les bombes logiques sont des virus conçus pour accomplir des dégâts lors de leur déclenchement par la satisfaction d'une condition: une date, une donnée particulière etc.

• Les vers ou *worm* sont des virus destiné à détériorer les données d'un système. Ils se reproduisent d'une façon exponentielle d'un ordinateur à un autre en se propageant en empruntant, comme support de transmission, les liaisons entre ordinateurs. Leurs objectifs est d'immobiliser le maximum de ressources de la machine qui demeurent difficilement utilisables.

• Les bactéries sont des programmes qui se multiplient d'eux-mêmes de façon exponentielle jusqu'à surcharger le système et dégrader sa performance.

I.2.4.2. Les différents remèdes

Plusieurs mesures peuvent annihiler ces attaques et violations de la sécurité des systèmes informatiques. Il y a ceux qui sont indiqués dans le début de ce chapitre, et qui sont plus générales, telles que les contrôles d'accès aux systèmes, aux machines ainsi qu'aux terminaux. Et d'autres, qui sont plus spécifiques, telles que:

- les contrôles d'accès aux données et aux informations sensibles. Il y a deux types de contrôles d'accès:

1. contrôle d'accès discrétionnaire (DAC: *Discretionnary Access Control*): dans ce cas, l'utilisateur est le seul responsable, c'est lui qui choisit le type de protection de ses fichiers;
2. contrôle d'accès mandaté (MAC: *Mandatory Access Control*): ici, les systèmes se chargent de la protection des fichiers. Ce mode de contrôle est adopté par des ordinateurs qui manipulent des données très sensibles;

- la cryptographie: elle consiste à encrypter les données au moyen d'un jeu de clés, de manière à les rendre illisibles à celui qui ne possède pas la clé. En général, l'encryptage est effectué par des algorithmes de codification; un algorithme inverse, permettra ensuite de reconstituer l'information originale. Parmi les algorithmes les plus répandus en cryptographie, on peut citer l'algorithme DES (*Data Encryption Standard*), RSA (*Revest, Shamir and Adelmann*);

- d'autres méthodes de sécurité sont utilisées telles que les sauvegardes, les procédures antiviraux, etc.;

- l'audit de la sécurité: cette technique consiste à enregistrer dans un ou plusieurs fichiers des informations captées, sur le réseau, sur l'activité des utilisateurs. Cette technique va être développé tout le long de ce chapitre.

I.2.4.3. Les différents critères d'évaluation

Les critères d'évaluation de la sécurité des systèmes informatiques définissent des moyens afin de contrer et de limiter ces menaces au bon fonctionnement des systèmes: ceux-ci ont parfois des conséquences considérables.

Il existe différentes normes de référence qui définissent des critères de classement en niveau de la sécurité d'un système informatique. Plusieurs organisations se sont occupé de fixer les critères pour ces classements. D'abord, le *Department of Defense* qui, en 1985, a mis au point les premiers critères dans leur livre "*Trusted Computer System Evaluation Criteria*" [DOD 83] aux Etats Unis, suivi par les allemands avec leur référence "*IT-Security Criteria*", la communauté Européenne "*Information Technology Security Evaluation Criteria*", et le Canada "*The Canadian Trusted Computer Product Evaluation Criteria*".

Tous les documents susmentionnés font référence à un document de base du nom d'"*Orange Book*" [DoD 83] dans lequel le DoD a défini 6 conditions fondamentales dérivées de l'objectif de base, dont 4 s'occupent du contrôle d'accès à l'information, et les deux autres assurent la crédibilité de l'accomplissement de ces contrôles dans le système. Ces conditions sont:

1. La politique de sécurité: il doit y avoir une politique de sécurité explicite et bien définie forcée par le système.
2. Les marques: chaque objet doit être associé à un "label" qui indique le niveau de sécurité de l'objet.
3. L'identification: chaque objet doit être identifié d'une façon unique et convaincante. Cette identification est nécessaire afin que la demande d'accès puisse être vérifiée.
4. La responsabilité: le système doit maintenir des actions complètes qui touchent à la sécurité. De telles actions comprennent l'introduction de nouveaux utilisateurs dans le système, changement du niveau de sécurité d'un objet ou d'un sujet et des tentatives d'accès non autorisés.
5. L'assurance: le système informatique doit contenir des mécanismes qui renforcent la sécurité, et il doit être possible d'évaluer l'efficacité de ces mécanismes.
6. La protection continue: les mécanismes qui implémentent la sécurité doivent être protégés contre les changements non autorisés.

Toutes ces exigences doivent être respectées par les systèmes. Les classes sont basées sur des divisions qui peuvent être divisées en catégories. Il existe 4 divisions principales: A, B, C et D. Dans chaque division, il y a des niveaux différents. Une description plus détaillée de ces niveaux est présentée dans ce qui suit:

⇨ Les niveaux de sécurité du DOD

Les quatre divisions de sécurité définis par le DOD se composent de niveaux ou classe(s). Les niveaux supérieurs doivent respecter les contraintes des niveaux inférieurs. Les critères ont été développés dans le but d'atteindre trois objectifs: (1) fournir aux utilisateurs un critère d'évaluation du degré de confiance d'un système informatique pour le traitement des informations sensibles; (2) fournir des conseils aux producteurs dans le but de satisfaire les exigences de sécurité pour les applications sensibles; et (3) fournir une base pour la spécification des exigences de sécurité dans les spécifications d'acquisition [DoD 83]. Ces niveaux sont définis dans l'"*Orange Book*" comme suit:

- Le niveau D est le niveau de protection minimale. Il n'y a pas de notion de protection entre utilisateur dans les systèmes de classe D (Ex. les micro-ordinateurs).

- Le niveau C représente la sécurité avec accès discrétionnaire (cf. I.2.1.3.3.). Ce niveau se découpe en deux classes:

  1. Classe C1: la sécurité à ce niveau est assez réduite. Les systèmes classés C1 ne sont pas suffisamment protégés contre les intrusions extérieures; l'environnement fourni sépare les utilisateurs de leurs données. Les utilisateurs doivent être autorisés de protéger leurs propres données pour ainsi limiter l'accès des autres utilisateurs ou groupes d'utilisateurs. Ceci suppose une authentification (procédure login/passwd) et un outil de protection fondé sur la notion de propriété (permission d'accès par utilisateur, groupe ou autres utilisateurs).

  2. Classe C2: dans laquelle l'authentification est plus stricte. Un système de cette classe est muni aussi d'un mécanisme d'audit. Les utilisateurs sont responsables de leurs actions. De plus, les ressources du système sont soumises à un contrôle d'accès et la documentation doit être complète. (Ex: le système *SunOS Release 4.1.*).Le niveau C2 exige simplement un mécanisme de génération d'information d'audit sans rien mentionner à propos de l'analyse de cette information.

- Le niveau B: La sécurité est mandatée (cf. I.2.1.3.3.). Cette division est découpée en trois classes:

  1. Classe B1: en plus des exigences du niveau C2, la classe B1 dans laquelle les objets contrôlés sont individuellement étiquetés, le système doit assurer le respect des étiquettes. Celles-ci ne sont pas modifiables par les utilisateurs. L'accès aux objets se base sur l'étiquette, celle-ci en détermine le droit d'accès. (Ex: *SunOS MLS Release 1.0*);

  2. Classe B2: dans laquelle: 1) la sécurité doit être représentée par un modèle; et 2) il doit être possible de tester les outils de sécurité disponibles. Le système doit être associé à une notion de privilège minimal. Un des rares systèmes classés au niveau B2 est le système MULTICS de *Honeywell Information System*;

3. Classe B3: est appelée "Domaines de sécurité". La structure interne du système doit être complètement documentée et prouvée informellement. Les listes de contrôle d'accès sont obligatoires. L'administration de la sécurité doit être une tâche séparée de l'administration du système lui-même. Dans cette classe, le système est très résistant aux tentatives d'intrusion.

- Le niveau A: La seule classe de la division A est la classe A1. Une preuve *formelle* est demandée. Il faut un modèle pour la protection du système et une preuve pour sa consistance et son adéquation, une spécification formelle de haut niveau du système de protection, une démonstration de la correspondance des spécifications de haut niveau au modèle, une implémentation informelle correspondant aux spécifications et une analyse formelle des canaux secrets.

I.2.5. La génération et l'analyse des traces

I.2.5.1. L'audit de sécurité

Il existe plusieurs types de préventions et de contre-mesures pour contrer les violations de sécurité. L'audit de sécurité est une méthode qui essaye de prévenir et détecter ces violations de sécurité. C'est une contre-mesure qui tente essentiellement de détecter toute activité pouvant nuire à la sécurité et éventuellement empêcher sa reproduction. Dans notre cas, cette technique permet de rassembler des informations qui circulent dans le réseau telles que:

- qui a essayé d'effectuer telles opérations;
- quelles sont les opérations effectuées sur le système;
- quels sont les opérations qui ont échouées un certains nombre de fois;
- de quel machine provient telle information;
- quelle est la destination et/ou l'origine de tel message ou information;
- qui a effectué des opérations anormales (*login failed* etc.);
- quel est le contenu du message transmis à partir de telle machine et les ports utilisés;
- etc.

Les systèmes équipés d'un mécanisme d'audit génèrent des fichiers de traces différents des "*Audit Trail*" généré par le système d'audit des commandes système SunOS relatives à la sécurité (objet d'un autre mémoire). Ces fichiers de traces captés sur le réseau forment une collection d'informations permettant de retracer certaines activitées sélectionnées.

Le fichier de traces est composé d'enregistrements appelés "audit records". Chaque audit record représente un paquet (cf. Annexe D.). Ces paquets peuvent être de différents types de protocoles: tcp, udp, arp etc. (cf. partie II). Ils constituent des évenements tels que un telnet, un login etc. Les chiffres hexadécimaux se trouvant dans le paquet donnent toutes les informations sur l'événement telles que, par exemple, la machine source et destination, le port source et destination, la donnée, le type de protocole utilisé dans la communication, la longueur de l'information, etc. Ces informations seront très utiles dans la sélection, l'analyse et la sécurité du système.

I.2.5.2. Les événements et la sélection

⇨ Les événements auditables

Tout événement lié à la sécurité du système est un événement auditable. D'après les recommandations du "*National Computer Security Center*", les événements à auditer dépendent du niveau de sécurité du système [Libion 91].

D'une façon générale un événement est représenté par l'action d'un sujet sur un objet, et ayant un certain résultat. Le sujet est une entité active qui reçoit ou qui agit sur l'information. Cependant, "l'objet est une entité passive qui contient ou reçoit de l'information" [DoD 83]. L'action qui est faite par le sujet sur l'objet est caractérisée par un résultat qui peut être un succès ou un échec de l'action.

⇨ La sélection des événements

Les systèmes munis d'un mécanisme d'audit peuvent générer une quantité énorme d'information. Ceci pose des problèmes d'espace disque et rend la tâche difficile à l'officier de sécurité. Cette tâche consiste à parcourir la masse de données afin de détecter la présence éventuelle d'anomalies. Pour remédier à cet inconvénient, le système générateur de fichiers de traces offre des possibilités de faire des présélections, via des paramètres, pour ne produire que certains événements en fonction des besoins de l'officier de sécurité (cf. Annexe C.). Toutefois, une présélection très stricte a le désavantage d'ignorer certains événements intéressants pour la sécurité. La présélection peut porter par exemple sur:

- le type de paquet à capter (TCP, UDP ou autres);
- le nombre de paquets à capter;
- la machine et/ou l'application (son adresse/numéro de port) à auditer;
- l'adresse source et/ou de destination des messages à capter;
- les ports source et/ou destination à surveiller;
- etc.

La présélection est appliquée au moment de la génération du fichier des traces. Nous pouvons ainsi capter les paquets, qui circulent entre deux machines données, et par la suite sélectionner, à l'aide des requêtes RUSSEL (cf. Annexe A.), ceux qui émettent ou reçoivent les messages sur des "ports" particuliers. Par exemple, surveiller le port n° 25 qui correspond au mail.

I.2.5.3. Les fichiers de traces

Les systèmes d'exploitation génèrent une description de quelques aspects sélectionnés sur les informations circulant dans le réseau dans un fichier de trace. Ainsi, l'analyse intelligente de ces fichiers constitue une tâche importante de la gestion de la sécurité du système.

Un fichier de trace est ainsi un ensemble de description présélectionné des événements de sécurité écrit par le mécanisme d'audit dans un ou plusieurs fichiers. Une description d'un événement de sécurité est composé d'un ensemble de données d'audit rassemblés dans un enregistrement. Une donnée audit est une information nécessaire pour décrire ou reconstituer un événement de sécurité. Cette donnée peut être un champ d'un enregistrement d'audit (Ex:

les caractères constituant le mot de passe introduit par l'utilisateur lors de sa connexion au système).

Après sa création, le fichier de traces peut être analysé: l'auditeur autorisé peut extraire les informations de sécurité sélectionnées. Pour cela, il faut exécuter la commande "etherfind" (cf. Annexe C.) avec les options appropriées. Cette commande a pour objectif de capter des paquets sur le support Ethernet (cf. I.1.1.). Elle est disponible dans le logiciel du réseau comme plusieurs autres options. Cependant, il faut disposer des privilèges nécessaires pour pouvoir l'exécuter. Sa syntaxe est la suivante:
etherfind [-d] [-n] [-p] [-r] [-t] [-u] [-v] [-x] [-c count] [-i interface] [-l length] expression .

Etherfind affiche à l'écran les informations qu'elle capte à partir des paquets qui passent sur le support Ethernet. Ces paquets sont ceux qui correspondent à l'expression booléenne expression. Selon l'option choisie, nous pouvons capter plusieurs types d'informations; les plus pertinentes se trouvent dans la zone de donnée du paquet analysé. L'option qui nous permette de capter ces données est l'option [-x]. Cette option permet d'ajouter dans le fichier de traces les informations qu'elle vient de capter. Ces informations seront écrites en mode hexadécimal. Cette option capte donc tous les paquets qui sont émis ou reçus d'une machine ou entre deux machines; les adresses de ces machines (leurs noms) sont spécifiées dans expression (dans l'Annexe C vous trouverez une description plus détaillée de cette commande).

Le fichier de trace est le résultat de cette commande. C'est un fichier séquentiel en mode ascii.

⇨ Protection des fichiers de traces

Des mesures très strictes vis-à-vis des fichiers de traces doivent être prises en considération. Le droit d'accès à ces fichiers ne peut être qu'en lecture seulement et par des personnes autorisées, cela afin de préserver l'intégrité des informations "audit data" qu'ils abritent. L'accès en écriture aux fichiers de traces n'est permis que par le mécanisme d'audit. Ce dernier est responsable de l'enregistrement de toute activité liée à la sécurité.

⇨ Format du fichier de trace

La commande etherfind avec l'option (-x) et la spécification des machines source et/ou destination, par exemple, donnent comme résultat un fichier ascii composé de paquets (cf. Annexe D.). Ces paquets représentent les informations nécessaires à l'envoi d'un message d'une machine à une ou plusieurs autres, ainsi que le contenu du message transmis. Les champs qui composent ces paquets sont spécifiés de façon approfondie dans la partie II de ce mémoire. Ces paquets sont représenté en mode hexadécimal (si on utilise l'option -x). Ce mode est le seul qui fournit une trace complète des informations captées. Les paquets peuvent être de deux types: TCP et/ou UDP (cf. II.5. et II.6.). Un exemple détaillé d'un fichier de trace est donné dans l'Annexe D.

Le but principale de l'audit est de garder la trace de toute interaction avec le réseau, permettant ainsi de découvrir toute action malicieuse et éventuellement d'empêcher sa reproduction. Les activités des utilisateurs sur le système sont enregistrées sous forme de records ou paquets par le mécanisme d'audit. Ces derniers doivent faire l'objet d'une analyse judicieuse et adéquate. Dans la section suivante, un outil d'analyse nouveau dans le domaine de la sécurité logique va être présenté. Il porte le nom d'"ASAX".

I.2.6. Asax: un outil d'analyse de traces

I.2.6.1. Introduction

Les systèmes informatiques actuels sont vulnérables aux différents types d'attaques (intrusion, virus, etc.). Il n'est donc pas possible de disposer d'une sécurité absolue, du niveau de sécurité A par exemple, comme nous ne pouvons pas prévoir toutes les menaces pouvant se présenter. La conception d'un système fiable contre toutes les attaques est difficile à réaliser actuellement car pour cela il faut donner une formalisation de tous les mécanismes d'accès permettant de prouver mathématiquement l'impossibilité d'une utilisation non autorisé ou accidentelle. Ce type de formalisation n'est pas encore possible à implémenter étant donné la complexité des systèmes d'exploitation et des protocoles de communication etc.

Les systèmes munis d'un haut niveau de sécurité comportent des mécanismes de stockage d'informations échangées avec le réseau dans des fichiers de traces. Ces fichiers seront analysés et donc la détection des attaques sera exécutée a posteriori. Cette analyse permettra de déceler des scénarios d'utilisations malicieuses, des tentatives d'intrusions, des virus etc. après leurs productions (ou même en cours de productions).

Le projet ASAX (*Advanced Security Audit-trail Analysis on uniX*) a pour objectif d'analyser ces fichiers de traces, il décrit un langage général pour de telles analyses. Ce langage est basé sur des règles et est appelé RUSSEL (*RUle-baSed Sequence Evaluation Language*).

Le logiciel ASAX, développé à l'institut d'Informatique en collaboration avec la société Siemens, est un outil qui permet l'analyse de n'importe quel fichier séquentiel en un seul passage. Il permet entre autre l'analyse des différents fichiers de traces générés par l'OS. Il fournit à l'utilisateur un langage d'interrogation simple qui lui permet d'exprimer ses requêtes d'analyse sur le fichier concerné. Cependant le fichier à analyser doit être mis sous un format standard "NADF" (cf. III.1.), simple et reconnu par ASAX.

Ce projet à pour but d'offrir des outils performants pour supporter ces analyses. Une des caractéristiques d'ASAX est son architecture qui est construite sur un outil d'analyse universel permettant l'adaptation du format de n'importe quel type de fichiers de traces. Une autre caractéristiques est le langage RUSSEL utilisé pour exprimer des requêtes sur ces fichiers. Ce langage est basé sur des règles qui sont bien adaptées à l'analyse des fichiers séquentiels en un seul passage.

Les outils d'analyse existants ne permettent pas d'analyser les fichiers de traces générés par les différents mécanismes d'audit. Ces mécanismes produisent des fichiers dont le format de l'enregistrement est différent du format exigé par l'outil d'analyse. ASAX a comme principale objectif de trouver des solutions à ces problèmes.

Pour être un outil puissant d'analyse, ASAX devrait affronter plusieurs types de problèmes liés à:

- la disparité des scénarios de violation de la sécurité;
- la grande quantité de données qui se trouve dans les fichiers de trace;
- La réutilisation des fichiers de traces et des outils d'analyses;

- La convivialité de l'interface fournit au gestionnaire de sécurité.

Parmi les différentes violations de sécurités rencontrés, nous pouvons citer les pénétrations externes et internes (cf. I.2.1.3.1.). Les violations externes sont caractérisées par des enregistrements répétés de "*login failed*" dans le fichier de traces, ce symptôme peut être détecté par une requête RUSSEL qui va parcourir le fichier séquentiel. Le second type d'intrusion concerne les utilisateurs autorisés à utiliser le système et qui essaient d'accéder, sans autorisation, à d'autres ressources. Nous pouvons ainsi classifier les approches de l'analyse des fichiers de traces en deux groupes:

- Les approches basées sur une statistiques d'une activité normale moyenne des utilisateurs, processus, ou de tout le système etc.

- Les approches basées sur les connaissances des experts concernant les prédictions d'intrusion.

Le premier type d'approches est approprié à la détection des scénarios d'intrusion connus, alors que le deuxième type est plus approprié pour détecter des scénarios inconnus.

L'outil d'analyse des fichiers de traces devient parfois un système expert dont le rôle est non seulement de détecter les violations de sécurité, mais aussi d'améliorer sa propre connaissance en mettant à jour le profil de quelques utilisateurs, et en ajoutant les scénarios de quelques nouvelles attaques.

Un autre facteur important qui rend l'analyse encore plus difficile est relié à la taille des fichiers de traces. Si nous ne faisons pas plusieurs filtrages importants et des sélections adéquates, le système d'exploitation pourrait produire une très grande quantité de données d'audit qui rend l'analyse impossible. Le système d'exploitation permet une sélection à partir de la source mais pour éviter la production de fichiers de traces inutilisables, cette sélection doit être très adéquate.

L'audit intelligent doit ainsi fournir une sélection adéquate et appropriée, cette sélection peut être exécutée au niveau de la source, dans ce cas elle est appelée une présélection, ou bien au niveau de l'analyse, on parlera alors d'une post-sélection. En pratique, une phase de filtrage de pré-traitement qui utilise des règles simples pourrait soulager l'outil d'analyse en écartant les grandes quantités de données non pertinentes. Dans le cas d'etherfind, la présélection est représenté par la spécification du nombre et du type de paquets à capter, de la machine cible et/ou destination à surveiller, etc.

Un autre problème concernant la quantité de données à analyser est relié au choix entre l'analyse "*on-line*" et l'analyse "*off-line*", dans notre cas c'est l'analyse "*off-line*" qui est exécutée. L'analyse est "*off-line*", si elle est exécutée après la création des fichiers de traces. Cependant, et sous certaines conditions, l'évaluation peut également s'effectuer pendant la création de ces fichiers, et nous parlerons d'analyse "*on-line*". Dans le cadre de ce mémoire, la faisabilité de ce type d'analyse nécessite un choix préalables des machines à surveiller, du type de paquets à capter, etc.

I.2.6.2. Les objectifs

Le but du projet ASAX est de définir et implémenter un système expert pour une analyse universelle, efficace et puissante des fichiers de traces, correspondant au niveau de sécurité B3 (cf. I.2.1.3.4.).

L'universalité se base sur son architecture. La réutilisation dans ASAX concerne le niveau physique. L'objectif principal étant de fournir un outil universel d'analyse des fichiers de traces qui tourne sur une grande gamme de systèmes d'exploitation. La raison de ce choix est double:

- permettre l'analyse de tous les fichiers de traces fournis et qui sont préalablement traduits en un format approprié;
- assurer le maximum d'efficacité qui reste l'objectif majeur de l'analyse des fichiers de traces.

En pratique, on définie un format de fichier d'audit normalisé (NADF pour *Normalized security Audit Data Format*) qui doit être assez flexible pour que tous les fichiers de traces existants peuvent être traduits facilement. L'analyse est ainsi exécutée sur seulement des fichiers de traces normalisés. Cet adaptateur de format fera l'objet du premier programme spécifié dans la partie III de ce mémoire. Il aura pour objectif de traduire chaque fichier de traces original ou fichier natif en un fichier binaire NADF.

I.2.6.3. Les problèmes principaux

Plusieurs problèmes que nous rencontrons pendant l'analyse peuvent être résolus. Le problème de la taille des fichiers de traces, par exemple, constitue une contrainte assez importante car la gestion de l'espace disque pourrait devenir critique pour l'officier de sécurité. Le temps CPU est aussi vulnérable à cette contrainte. Deux approches peuvent être retenues pour faire face à cet inconvénient:

- La présélection;
- La compression du donnée.

L'objectif étant donc de réduire la taille de ces fichiers de traces, cependant ceux-ci restent suffisamment importants: nous ne pouvons pas dire a priori quelles sont les informations qui vont être utilisées pour l'analyse. Ce problème peut être résolu selon que nous choisissons un nombre raisonnable de paquets à capter, ou que l'on opte pour une analyse *on-line* avec des options bien appropriées.

D'autres problèmes persistent. Ce sont par exemple:

- La variété de scénarios dans la sécurité;
- La réutilisabilité de l'outil de l'analyse;
- L'interface utilisateur.

I.2.6.3.1. La variété de scénarios dans la sécurité

Les scénarios dans la sécurité des systèmes informatiques prennent diverses formes (intrusions, virus, etc.) La classification des différentes formes de menaces par des chercheurs a pour but de définir des catégories de scénarios détectables par l'évaluation des fichiers de

traces. Néanmoins, dans chaque classe, beaucoup de scénarios d'attaques sont possibles, dont certains restent toujours imprévus. La difficulté provient donc du fait que l'on doit connaître a priori un scénario d'attaque et interroger le fichier de trace normalisé. Cela suppose une expertise non seulement des scénarios d'attaques mais aussi de la façon dont ils sont représentés dans le fichier.

I.2.6.3.2. La réutilisabilité, la généricité et l'universalité

La réutilisabilité d'un logiciel est une qualité très recommandée. Un logiciel peut être réutilisé dans différents environnements, matériels et logiciels, avec un effort d'adaptation minimal. Dans le cas d'outil d'analyse de fichiers de traces cette qualité est très importante vue la diversité des systèmes d'exploitation et de leurs versions multiples. Deux approches sont à envisager pour atteindre la réutilisabilité:

1. la conception d'un outil d'analyse paramétrisé et dans lequel les formats des différents fichiers de traces peuvent être instanciés. Il s'agit d'un outil "générique";

2. ou, un outil plus général permettant d'analyser tout type de fichiers de traces après conversion préalable en un format approprié à l'outil d'analyse: nous parlerons d'outil "universel".

I.2.6.3.3. L'interface utilisateur

L'analyse des fichiers de traces présente un autre problème lié au confort d'utilisation de l'évaluateur: en effet, l'ergonomie de l'interface utilisateur est primordiale. L'outil d'analyse doit être convivial dans le sens où il permet à l'officier de sécurité d'effectuer ses analyses sans aucune difficulté.

I.2.6.4. Les qualités d'ASAX

ASAX est un outil de sécurité. Il permet d'analyser tout fichier séquentiel. Il a été conçu pour atteindre les objectifs suivants:

- analyser en un seul passage de longs fichiers séquentiels de sécurité des systèmes, mais aussi le traitement d'autres applications tels que l'analyse de flux de données via un réseau, etc.

- accepter toutes requêtes d'analyse possible y compris les requêtes qui portent sur la relation entre plusieurs enregistrements.

Les caractéristiques principales de l'évaluateur ASAX sont: l'**universalité**, la **puissance**, l'**efficacité** et la **portabilité**. Elles sont développées dans les sections suivantes.

I.2.6.4.1. L'universalité

Les systèmes d'exploitation munis d'un mécanisme d'audit génèrent des fichiers de traces de formats différents. L'objectif est de fournir un outil d'analyse universel qui pourrait tourner sur une large gamme de systèmes d'exploitation. Cette universalité est réalisée par le choix et la définition d'une structure de fichier souple et standardisée, appelé NADF.

L'analyse s'effectue uniquement sur des fichiers aux formats standardisées. Les fichiers de traces doivent donc être traduits en fichier de format NADF (cf. Chapitre III.1.).

Vu la flexibilité du format NADF, la conversion de tout type de fichiers de traces en un tel format se ferait plus facilement avec la réalisation d'un adaptateur de format. Un des objectifs de ce mémoire est d'implémenter un adaptateur de format pour les fichiers de traces générés par le système. Le chapitre III.1. traitera de la mise en oeuvre d'un tel adaptateur de format.

En pratique, le format NADF est suffisamment flexible que tous les fichiers de traces existants peuvent être traduits de façon simple. L'analyse de ces fichiers de traces est alors exécutée sur les fichiers normalisés. L'adaptateur de format devrait être fourni pour traduire chaque fichier de traces en un fichier de format NADF.

La faisabilité d'une telle approche a été démontrée par l'implémentation de formats adaptateurs pour les fichiers de traces générés par les systèmes d'exploitation BS2000 et SINIX de Siemens.

Une architecture simplifiée d'ASAX est présentée dans la figure 2.2.4.1.

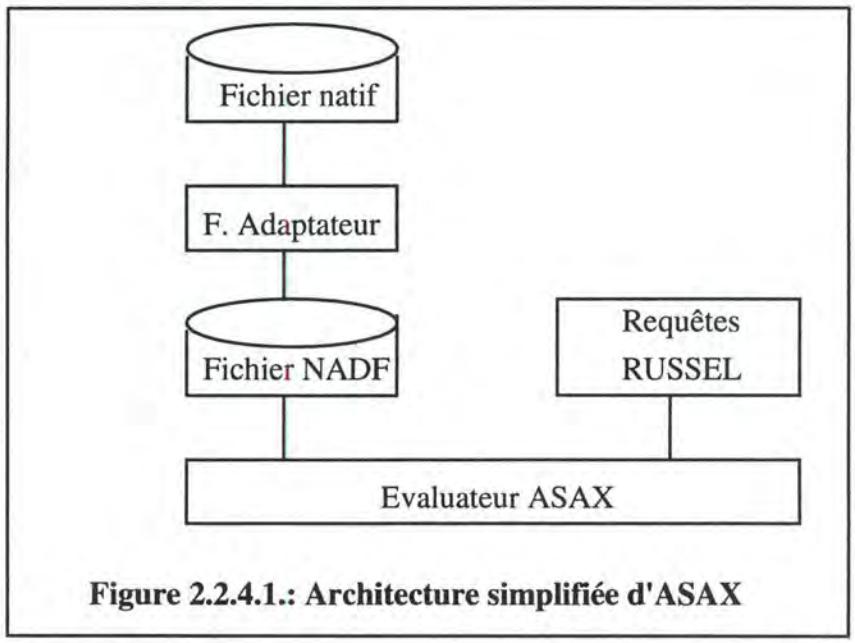


Figure 2.2.4.1.: Architecture simplifiée d'ASAX

Le simple format des fichiers NADF garantit la portabilité au niveau physique. L'expérience avec BS2000 et SINIX montre que l'implémentation de tels adaptateurs de formats est assez simple.

I.2.6.4.2. La puissance: Le langage RUSSEL

La puissance d'ASAX repose sur le langage de règles appelé RUSSEL (*RUle-baSed Sequential Evaluation Language*) qui permet à l'utilisateur d'exprimer ses requêtes d'analyse. Il s'inspire du principe des langages d'intelligence artificielle basé sur des règles avec chaînage avant. Plusieurs systèmes experts de sécurité utilisent l'approche "langage de règles" (*rule-*

based language), comme par exemple le système IDES [Lunt 90], système basé sur le modèle IDES, il utilise PBEST: un outil expert général basé sur des règles.

RUSSEL est un langage de règles conçu spécifiquement pour l'analyse des fichiers de traces, et d'une façon plus générale, pour celle de n'importe quel fichiers séquentiel. Ce langage sera utilisé pour reconnaître des champs particuliers dans les fichiers séquentiels et ainsi déclencher des actions appropriées telles que l'activation d'une alarme, l'envoi d'un message etc. L'idée de RUSSEL est de prendre l'avantage de cette utilisation particulière et rendre le langage aussi efficace et facile à utiliser que possible.

RUSSEL peut être considéré comme un langage procédurale qui emploi des structures de contrôle pré-définies qui soient adaptés au raisonnement sur les séquences d'enregistrement. Ces structures de contrôle sont basés sur un mécanisme de déclenchement de règles qui peut être résumé comme suit:

- le fichier de traces normalisé ou l'NADF est analysé enregistrement par enregistrement d'une façon séquentielle au moyen d'un ensemble de règles. A chaque instant, l'enregistrement courant est analysé et l'ensemble des règles est actif;

- la règle active encapsule les connaissances essentielles concernant le passé de l'analyse. Cette connaissance est par la suite fournie à l'enregistrement courant sur lequel nous allons exécuter les règles. Le traitement génère à son tour des nouvelles règles qui vont être appliquées ultérieurement;

- selon le point de vue du programmeur, une règle est considérée comme une procédure paramètrée classique mais qui peut entraîner des déclarations d'un type particulier d'actions: le déclenchement de règles;

- pour déclencher une règle il faut fournir les paramètres réel pour la règle et spécifier le moment de déclenchement de cette règle: pour l'enregistrement courant, l'enregistrement suivant ou pour la fin de l'analyse;

- le processus est initialisé par un ensemble de règles activé pour le premier enregistrement.

Pour programmer une requête dans le langage RUSSEL, il faut écrire un nombre de règles adéquates qui seront activer conformément au plan de contrôle général indiqué ci-dessus. Ces règles respecte une syntaxe particulière. Cette syntaxe et la sémantique du langage RUSSEL est présenté dans l'Annexe B.

Le fait que les fichiers de traces sont triés de façon chronologique joue un rôle essentiel dans l'approche qui stipule que l'analyse de ces fichiers consiste généralement en une recherche des sous-séquences chronologiques parmi tous les enregistrements d'événements.

Une déclaration de règles dans le langage RUSSEL consiste en un nom, un ensemble de paramètres formels, un ensemble de variables locales et une partie action. Une règle active est une instantiation d'une déclaration de règle avec des paramètres réels. Les règles actives sont exécutées une à la fois (à tout instant, il n'y a qu'une seule règle exécutée). Les structures de contrôle classique peuvent être utilisées pour écrire des actions conditionnelles, des actions répétitives, des actions séquentielles, etc. Les actions élémentaires comprennent des

affectations, des déclenchements de règles et des routines d'appel externes. La dernière action permet au programmeur d'utiliser des procédures écrites dans un langage externe, comme le langage C, pour la lecture, l'écriture, l'accès aux structures de données complexes, etc.; les routines externes peuvent être appelées pour exécuter tous les caractéristiques qui ne concernent pas le traitement séquentiel du fichier de traces (pour de plus amples détails cf. Annexe A et B).

Les actions sont généralement appliquées sur quelques enregistrements spécifiques (Ex: ceux correspondant à des événements donnés, concernant un sujet donné, etc.).

Une règle est explicitement redéclenchée pour analyser l'enregistrement suivant, par défaut elle meurt après son déclenchement. L'objectif étant d'avoir un contrôle simple et directe sur la durée de vie des règles et pour éviter des mécanismes complexes qui tuent les règles dépassées.

La présentation du langage RUSSEL ci-dessus montre que ce langage est assez puissant pour supporter toutes les requêtes nécessaires qui concernent les fichiers séquentielles des traces. Cependant, le langage peut paraître difficile comme il utilise des styles différents de programmation. Le programmeur devrait donc raisonner en tenant compte de la structure de contrôle entraînée par le mécanisme du déclenchement des règles. Cette complexité semble être acceptable pour deux raisons:

- En comparaison avec d'autres langages moins efficace, RUSSEL ne semble pas être plus difficile dans son utilisation. Dans le langage RUSSEL, une approche pragmatique est considérée: l'efficacité des langages procéduraux qui est reconnu est bien exploitée. Cependant, les règles peuvent être écrites ou lues de façon déclarative. Nous pouvons ainsi dire que ce langage offre un bon compromis entre l'efficacité et l'aspect déclaratif de ses règles.

- L'idée principale du projet ASAX est de fournir un langage puissant, portable et implémentable de façon très efficace. Cependant, RUSSEL ne peut pas être considéré comme le langage final de l'utilisateur avec lequel les responsables de sécurité pourront construire leurs requêtes, mais seulement comme un langage intermédiaire. Entre autre, une librairie de règles de sélection prédéfinies de haut niveau sera définie. De plus, il est prévu de concevoir un langage de haut niveau, que nous appellerons RUSSEL2, qui permettrait l'expression des requêtes dans un modèle de haut niveau (utilisant par exemple des objets et des sujets) et dans un style plus déclaratif. Les requêtes dans le langage RUSSEL2 seront automatiquement traduites en langage RUSSEL. Le langage de haut niveau (RUSSEL2) sera plus facile à utiliser que RUSSEL mais moins puissant; ainsi, quelques requêtes spécifiques continueront à être programmées directement dans le langage RUSSEL intermédiaire.

I.2.6.4.3. L'efficacité: l'implémentation

L'efficacité d'ASAX se base sur son implémentation. Cette efficacité est un aspect critique de l'analyse des fichiers de traces à cause de la grande quantité de données à analyser. L'efficacité nécessaire est accomplie par deux conceptions principales d'ASAX:

1. L'analyse du fichier de traces standardisé se fait en un seul passage séquentiel du fichier. A chaque instant, toute information sur le passé de l'analyse (c-à-d les enregistrements déjà analysés) est encapsulée dans l'ensemble des règles actives. Cette information est représentée par la valeur des paramètres effectifs de ces règles;

2. Le traitement d'un enregistrement courant est précédé d'un pré-traitement (pour une raison d'optimisation du temps d'accès) permettant un accès direct (via une table d'indirections) à ses champs lors de l'évaluation de la condition de sélection. Après avoir évalué le record courant, un ensemble de règles actives est appliqué à l'enregistrement suivant;

3. Evaluation efficace des conditions: les conditions qui figurent dans les règles sont évaluées de façon très efficace. L'évaluateur utilise une technique permettant de tronquer l'évaluation d'une condition dès que la valeur de vérité de celle-ci n'est pas vérifiée, sans devoir évaluer l'entièreté de la condition. Une telle efficacité d'évaluation des conditions est indispensable: cela correspond à une grande partie du temps CPU nécessaire à l'analyse.

Le premier principe est rencontré par le langage RUSSEL qui est spécialement conçu dans le but de permettre à toutes les requêtes d'être traitées en un seul passage. L'information nécessaire concernant le passé de l'enregistrement courant est encapsulée dans un ensemble de règles actives. En plus, comme RUSSEL supporte des appels de routine externe, les informations concernant le passé peuvent être également rangées dans les structures de donnée C et accédées via les routines externes.

Le second principe est accompli par les moyens des techniques efficaces d'implémentation qui se chargent de l'optimisation exigée. Quelques descriptions de cette implémentation sont mentionnées ci-dessous:

- Les règles RUSSEL sont traduites dans un code interne optimisé qui peut être traité de façon efficace par une machine appropriée. Une idée de l'optimisation à ce niveau consiste en la transformation des conditions booléennes qui sont représentées comme des arbres "et/ou" en des arbres binaires plus appropriés dans lesquels chaque noeud comprend une condition élémentaire, Cond, qui va être évaluée, et deux pointeurs: un pointeur pointe vers la condition suivante qui va être évaluée si Cond est vrai et le deuxième pointeur pointe vers la condition qui suit qui va être évaluée si Cond est faux. De cette façon, les conditions les plus élémentaires ne sont pas évaluées dans plusieurs cas. Comme les règles comprennent principalement des formes conditionnelles: "Condition->Action", cette technique peut être très satisfaisante.

- Comme le système va évaluer les règles qui aurait besoin des différents champs de données de l'enregistrement courant, un accès efficace à l'enregistrement courant est essentiel car l'enregistrement courant a une longueur variable et un format libre. Ainsi, avant d'appliquer les règles actives sur l'enregistrement courant, l'enregistrement est traité une fois dans le but de créer une table d'indirection optimisée qui contient de pointeurs vers les différentes données de l'enregistrement. Pendant le traitement de l'enregistrement courant, la donnée va être accédée via cette table de direction. La table d'indirection est mise dans une zone fixe au début. Elle est mise à jour chaque fois qu'un enregistrement devient courant. Ainsi, chaque référence à une donnée dans le code

interne de la règle pourrait être représentée par la localisation de l'entrée correspondante dans la table d'indirection, de cette façon toutes les données d'audit nécessaires sont disponibles pendant le traitement de l'enregistrement courant. Ceci minimise le temps d'accès pendant le traitement de l'enregistrement courant, toutes les références ou données seront des adresses absolues.

- Le système préserve à chaque fois trois ensembles de règles: celles qui sont actives pour l'enregistrement courant, celles qui vont être activées pour le suivant et celles qui vont être activées à la fin du traitement du fichier séquentiel. Chaque ensemble est représenté par une liste liée, dont chaque cellule contient deux pièces d'information: un pointeur vers le code interne de la règle et une liste des valeurs des paramètres réelles. Il faut noter que plusieurs instances de la même règle peuvent exister et utilisent toutes le même code interne. Passer de l'enregistrement courant au suivant se fait de façon simple en se déplaçant de l'ensemble des règles courantes vers l'ensemble des règles suivantes et par la réinitialisation de l'ensemble des règles suivantes vers l'ensemble vide.

- Quand le traitement d'une règle commence, les paramètres réels sont copiés dans un champs de rangement qui est le même pour toutes les règles. Ceci permet l'accès direct aux valeurs des paramètres réels.

I.2.6.4.4. La portabilité

L'évaluateur est implémenté dans un langage de haut niveau (le langage C), ce qui le rend facile à porter sur d'autres machines. En effet, la portabilité de cet analyseur est effective sur les systèmes d'exploitation Sinix, Unix et Dos. Actuellement, ASAX tourne avec succès sur MX2, MX300, Sun Sparc (dans le cadre du mémoire, la portabilité d'ASAX sur les Sun s'est accomplie sans difficulté majeure) et aussi bien sur PC DOS.

I.2.6.4.5. Autres caractéristiques d'ASAX:

ASAX va fournir un adaptateur de format paramètrisé qui réalise la fonction de l'adaptateur du format, en commençant par une description déclarative du format du fichier de traces natif. Il est à noter que l'architecture ASAX ne fait pas de suppositions préalables sur le choix d'une analyse "*on-line*" ou "*off-line*".

Partie II: Le modèle TCP/IP

Chapitre II.1

LE MODELE TCP/IP: Définition et architecture

II.1.1. Introduction

II.1.1.1. Définition générale

Le modèle ou la technologie TCP/IP (*Transmission Control Protocol, Internet protocol*) est le résultat des études de recherches financées par la DARPA (*Defense Advanced Research Projects Agency*). Cette technologie, d'après [Comer 88], comprend un ensemble de standards de réseaux qui spécifient les détails de communication entre ordinateurs ainsi qu'un ensemble de conventions qui régissent l'interconnexion des réseaux et le routage du trafic.

Cette technologie est appelée communément TCP/IP car le coeur de ce modèle est formé par le protocole IP (*Internet Protocol*) et par le protocole TCP (*Transmission control protocol*). Le protocole IP fournit un service de communication non fiable entre ordinateurs de l'Internet, il correspond assez bien à la couche 3 du modèle OSI. Le protocole TCP rend fiable ce service de communication, il est proche du protocole de la classe 4 de la couche transport dans l'architecture OSI.

Ces protocoles fournissent des recettes pour faire passer des messages dans le réseau, ils spécifient les détails du format des messages et décrivent des moyens pour corriger les erreurs éventuelles.

TCP/IP forme une technologie de base pour le réseau Internet (inter-connexion de réseaux) qui connecte la majorité des institutions de recherches aux Etats-Unis d'Amérique. La NASA (*National Aeronautics and Space Administration*), le NSF (*National Science Foundation*) et le département de l'énergie font partie de ces institutions. Ils utilisent TCP/IP pour connecter leurs sites de recherches avec ceux de la DARPA. La structure résultante est appelée l'Internet DARPA ou l'Internet TCP/IP ou encore l'Internet. Cette connexion permet aux chercheurs de ces institutions de partager les informations avec leurs collègues à travers tout le pays comme s'ils étaient tous dans le même bureau.

II.1.1.2. Les objectifs

Le modèle TCP/IP a été conçu pour connecter différents types de support de communication utilisant la technique du paquet. Ces supports sont reliés entre eux par des

passerelles. Ensemble, ils forment un réseau appelé l'Internet. Le modèle TCP/IP fournit un service fiable de communication entre processus s'exécutant sur des ordinateurs hôtes attachés à l'Internet.

Les réseaux de communication les plus développés dans les systèmes Unix sont basés sur la suite TCP/IP de l'Internet. Les liaisons physiques locales sont réalisées à partir de réseaux Ethernet pour la plupart (ce qui est le cas de l'Institut d'Informatique de Namur).

II.1.1.3. Historique

Au début des années 80, le département de la défense américaine (*DoD*) a mis au point les protocoles TCP/IP, précurseur des modèles basés sur les couches de communication. Ces protocoles de communication sont le fruit d'un projet, lancé au début des années 70, destiné à standardiser les moyens de communication entre les ordinateurs de l'administration de la défense américaine. Le groupe de recherche qui a mené ce projet était le premier à introduire le concept de "couches de communication". Le résultat de ce travail était l'élaboration des protocoles du réseau ARPANET. Celui-ci était suivi par le développement de TCP/IP dont les premières implémentations ont vu le jour en 1980. Cette série de protocoles a bénéficiée des innovations du moment, tels les réseaux à commutation de paquets, les réseaux à collision ainsi que diverses technologies devenues opérationnelles. L'université de Berkeley, les a intégrée dans son Unix BSD favorisant ainsi la généralisation de son utilisation. Il n'en reste pas moins que TCP/IP était le précurseur des modèles basés sur les "couches de communication", modèle que le standard OSI a plus tard consacré. Mais TCP/IP reste le plus ancien et aussi le plus simple.

Au début des années 90, vu le nombre croissant de réseaux, un nouveau problème se présente pour TCP/IP : son espace d'adressage est limité. Cependant, le modèle OSI offre une plus grande souplesse. Malgré ceci, le modèle TCP/IP est toujours relativement utilisé, tandis que le modèle OSI se répand de plus en plus.

Dans ce qui suit, nous parlerons d'abord des protocoles en générale; ensuite de leurs fonctions d'une façon plus approfondie; puis nous présenterons l'architecture conceptuelle du modèle et de son organisation en couches, nous évoquerons sommairement l'architecture en couches des modèle OSI et TCP/IP et enfin nous ferons une comparaison des deux.

II.1.2. Principes et fonctionnement de TCP/IP

II.1.2.1. Les protocoles: Objectifs et définitions

Les protocoles nous permettent de spécifier et de comprendre la communication sans chercher à connaître les détails d'un vendeur de réseaux particulier. Ils sont à la communication ce que sont les langages de programmation pour l'informatique. Par exemple, les détails de format d'une structure Ethernet, la politique d'accès au réseau, et la manipulation des trames d'erreurs sont pris en charge par un protocole qui décrit les communications physiques sur un réseau supportant le protocole Ethernet (cf. I.1.). D'une façon similaire, les détails des adresses Internet, du format du datagramme Internet, du concept de non fiabilité, et de livraison sans connexion sont compris dans le protocole Internet.

Les systèmes complexes de communication de données utilisent plus d'un protocole pour manipuler toutes les tâches de transmission, ils ont besoin d'un ensemble de protocoles coopératifs, appelés familles de protocoles. L'organisation d'une famille de protocoles est décrite en II.1.2.2.

Les problèmes qui peuvent surgir quand les machines communiquent entre elles sur un réseau de données sont de plusieurs types, les principaux sont:

- **Une panne du matériel:** un hôte ou une passerelle peut tomber en panne: le matériel se casse ou le système d'exploitation ne fonctionne plus. Une liaison du réseau de transmission peut tomber en panne ou peut être déconnectée accidentellement. Le système doit détecter une telle panne et si possible la réparer.

- **Un encombrement du réseau:** même si le matériel ainsi que le logiciel fonctionnent correctement, les réseaux ont une capacité finie qui ne peut être dépassée. Le système doit gérer l'encombrement pour que le trafic reste fluide.

- **Perte ou retard des paquets:** Parfois, les paquets sont perdus ou arrivent en retard. Le logiciel du protocole doit connaître les raisons de la perte ou s'adapte aux retards.

- **Une donnée corrompue:** les interférences magnétiques ou électriques peuvent causer des erreurs de transmission qui vont corrompre les contenus des données transmis. Le système doit détecter et corriger de telles erreurs.

- **Des erreurs de duplication ou de séquence:** les réseaux qui offrent des routes multiples peuvent délivrer les données en désordre. Le système doit ré-ordonner les paquets à l'arrivée et supprimer les doubles.

Avec tous ces problèmes et vu la diversité des tâches, il devient difficile d'écrire un seul protocole qui peut gérer le tout.

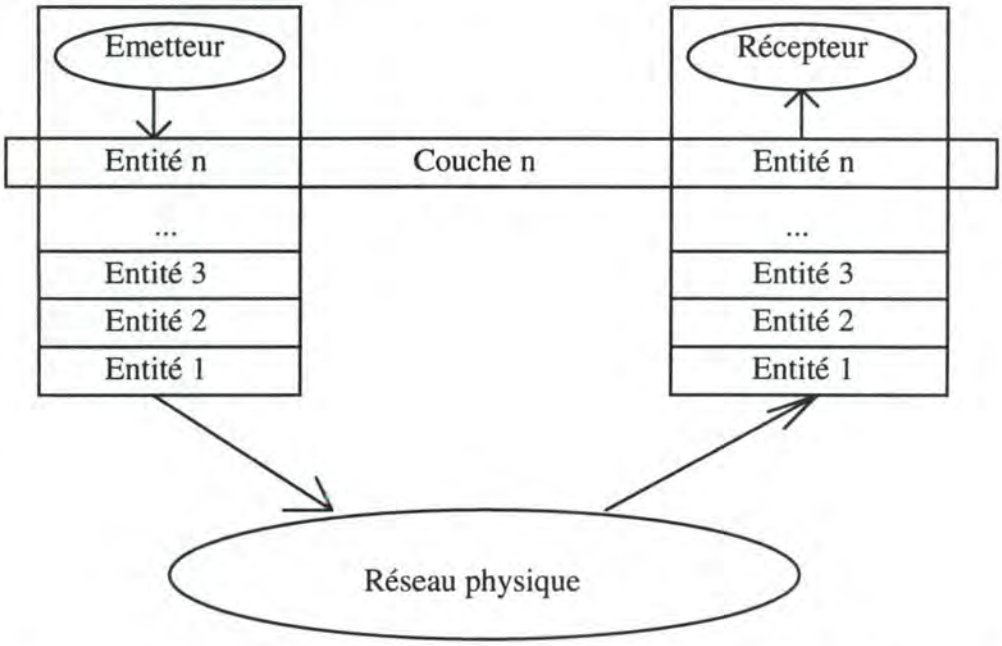
Par analogie aux langages de programmation, la division en sous problèmes donne au concepteur la possibilité de se concentrer sur une chose à la fois, et à celui qui fait l'implémentation de construire et tester chaque partie du logiciel indépendamment.

II.1.2.2. Architecture d'une famille de protocoles: les couches conceptuelles

Sur chaque machine, les modules du logiciel du protocole (le système) sont empilés verticalement en forme de couches. Chaque couche est responsable de la manipulation d'une partie du problème. Entre les couches, il y a une relation "utilise" qui a pour objectif de cacher les détails à l'utilisateur.

Généralement, transmettre un message d'un programme d'application sur une machine à un programme d'application sur une autre machine signifie que le message passe de haut en bas à travers les entités successives du logiciel du protocole de la machine émettrice, puis passe dans le réseau, et enfin remonte de bas en haut à travers les entités successives du logiciel du protocole de la machine réceptrice.

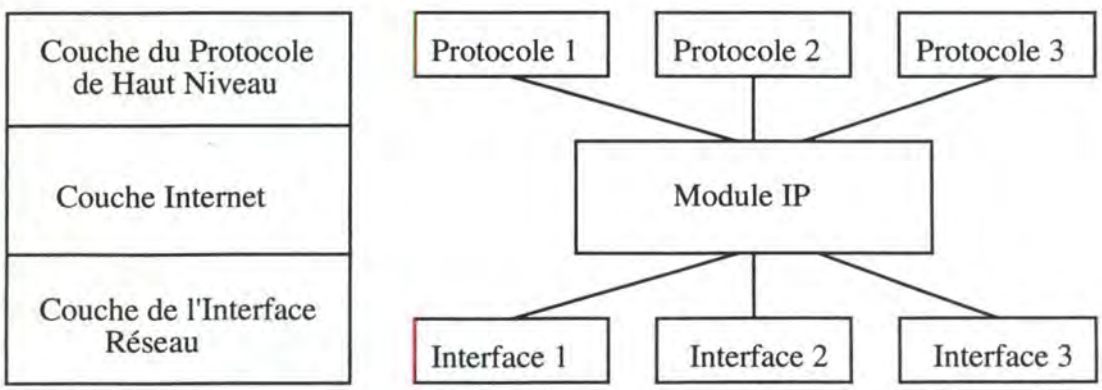
Ci-dessous une figure de l'organisation du logiciel en couches de familles de protocoles:



Les logiciels qui implémentent une famille de protocoles sont très complexes. Chaque couche prend des décisions dans la correction du message et choisit une action appropriée basée sur le type de message ou sur l'adresse de destination. Par exemple, une couche sur la machine réceptrice doit décider de garder le message ou de le réexpédier vers une autre machine. Une autre couche doit décider quelle application devrait recevoir le message, etc.

La différence entre l'organisation conceptuelle du logiciel du protocole et les détails d'implémentation est que dans le diagramme conceptuel, comme le montre la figure ci-dessous, on considère une couche appelée Internet entre les entités de protocole de haut niveau et l'entité de l'interface réseau, par contre le diagramme réaliste considère que le logiciel IP peut communiquer avec plusieurs entités (modules) de protocoles de haut niveau et avec plusieurs interfaces réseau.

Couches Conceptuelles **Organisation du Software**

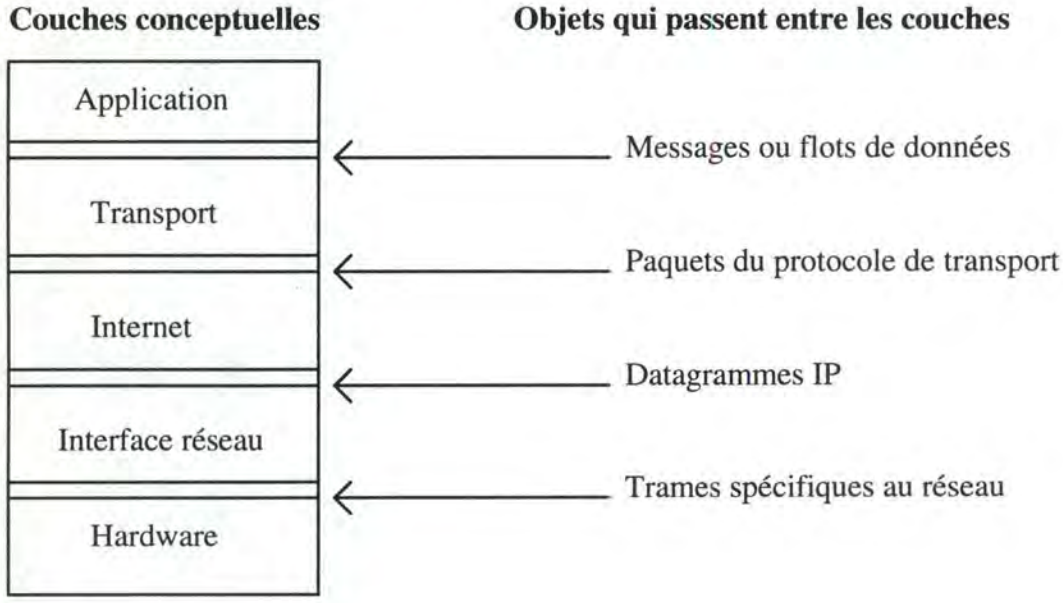


En fait, le modèle conceptuel a pour but de décrire les services offerts par chaque couche. Il ne décrit pas leur fonctionnement.

II.1.2.3. L'architecture en couches du modèle TCP/IP

Une fois la décision de la partition des problèmes de communication en sous-problèmes et de l'organisation du logiciel du protocole en modules est prise, chaque module va s'occuper d'un problème particulier et va occuper un emplacement précis appelé couche. La structure en couches va être utilisée ultérieurement par le modèle OSI.

Le logiciel de TCP/IP est organisé en quatre couches conceptuelles qui sont construites sur une cinquième couche du matériel (*hardware*). La figure ci-dessous montre les couches conceptuelles ainsi que les formes de données qui passent entre ces couches [Comer 88].



1. La couche interface réseau:

La couche la plus basse du logiciel du protocole Internet est la couche d'interface réseau. Cette couche reçoit des datagrammes IP et les transmet sur un réseau spécifique (sous-réseau). Une interface réseau peut être composée d'un dispositif conducteur ou un sous-système complexe qui utilise son propre protocole de liaison de données.

2. La couche Internet:

La principale mission de cette couche est d'obtenir l'interconnexion des différents sous-réseaux. Pour y arriver, un mécanisme unique de transport de données a été choisi. Ce mécanisme est le protocole IP. Le protocole IP regroupe toutes les fonctions nécessaires pour transmettre une suite d'octets, appelée datagramme, entre un ordinateur hôte source et un ordinateur hôte destination connectés éventuellement à des sous-réseaux distincts. Il ne dispose, toutefois, pas de mécanismes qui augmentent la fiabilité de la transmission des données de bout en bout, ou qui contrôlent le flux des datagrammes, ou encore qui règlent le séquencement des datagrammes. Le service offert par le protocole IP est donc non fiable.

Le protocole IP implémente trois fonctions de base : l'adressage, le routage et la fragmentation. L'adresse Internet est une adresse de longueur fixe. Elle permet d'identifier les ordinateurs hôtes et les passerelles de l'Internet. Cette adresse Internet est utilisée pour

acheminer les datagrammes à leur destination. Quand un datagramme doit traverser un sous-réseau dont la taille maximale du champ des données des paquets est inférieure à la taille du datagramme, le protocole IP fragmente le datagramme en fragments de taille plus petites. A l'arrivée, le datagramme initial est "reconstruit" avant d'être fourni à l'utilisateur. Le protocole IP traite chaque nouveau datagramme de manière indépendante des autres datagrammes. Ils pourront donc suivre des itinéraires différents avant d'arriver à leur destination.

La couche IP s'occupe donc des communications entre machines. L'entité IP de la machine source accepte une demande d'envoi d'un paquet d'une couche transport avec l'identification de la machine à laquelle le paquet devrait être transmis. Elle encapsule le paquet dans un datagramme IP, remplit l'en-tête du datagramme, utilise l'algorithme de routage pour déterminer si elle doit le livrer directement ou l'envoyer à une passerelle, elle passe le datagramme à l'interface du réseau appropriée pour la transmission. L'entité IP de la machine de destination s'occupe des datagrammes reçus, elle vérifie leur validité, supprime l'en-tête, et utilise l'algorithme de routage pour décider si le datagramme devrait être traité localement ou réexpédié. Pour les datagrammes destinées à une machine située sur le même sous-réseau que la passerelle, le logiciel de la couche Internet choisit parmi plusieurs protocoles de transport celui qui va recevoir le paquet.

L'encapsulation signifie que TCP, par exemple, crée un en-tête pour la donnée que l'utilisateur désire transmettre, ce datagramme est passé à la couche IP. La couche IP crée également un en-tête de ce qu'elle reçoit de TCP. Finalement, la couche interface du réseau met le datagramme dans une trame avant de le transmettre d'une machine à une autre. Le format de la trame dépend de la technologie du réseau utilisé.

A l'entrée, un paquet arrive à la couche la plus basse du logiciel du réseau et commence son ascension à travers les couches successive. Chaque couche enlève un en-tête avant de passer le message à la couche suivante de façon à ce que la couche la plus haute passe seulement les données au processus récepteur, sans aucun en-tête. Ainsi, l'en-tête le plus externe correspond à la plus basse couche du protocole, alors que l'en-tête le plus interne correspond à la plus haute couche du protocole.

Enfin, la couche Internet envoie des messages ICMP (*Internet Control Message Protocol*) et s'occupe de tous les messages ICMP reçus. Ce protocole (ICMP) permet d'envoyer des messages de contrôle et des messages d'erreur entre sites de l'Internet. Il est également chargé de renseigner l'ordinateur hôte source des problèmes qui surviennent aux datagrammes lors de leur transmission. Il peut également fournir à l'ordinateur hôte source des conseils ou des renseignements concernant le fonctionnement de l'Internet. Le protocole ICMP ne garantit toutefois pas que, si le datagramme n'a pas été livré à l'ordinateur hôte de destination, l'ordinateur hôte source en sera informé. Seuls les problèmes détectés par le protocole IP et pour lesquels un message ICMP est prévu, seront communiqués à l'entité IP émettrice.

3. la couche transport:

La première fonction de la couche transport est de fournir une communication entre deux programmes d'application situés sur des machines hôtes distinctes, au travers du réseau. Une telle communication est souvent appelée "de bout en bout". La couche de transport peut régulariser le flux d'information, elle peut également fournir un service de transport fiable, assurant une arrivée de données sans erreurs et dans l'ordre. Pour ce faire, elle reçoit des

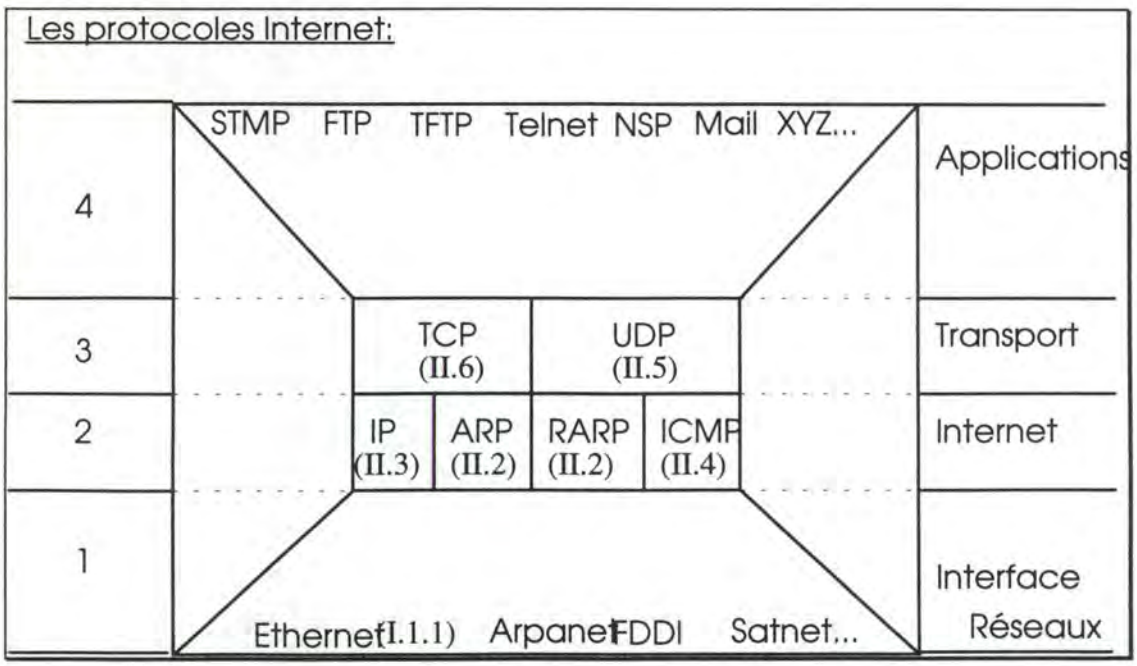
accusés de réception de la machine de destination et retransmet les paquets perdus. Le logiciel de transport divise le flot de données qui va être transmis en des petites pièces appelés paquets (de la terminologie OSI). Chaque paquet est transmis avec son adresse de destination au niveau suivant.

Dans la figure des couches conceptuelles on utilise un seul bloc pour représenter la couche application. Cependant, un ordinateur puissant peut avoir plusieurs programmes d'application qui peuvent communiquer avec d'autres sur machines différentes, via le réseau, en même temps. La couche transport doit accepter les données de plusieurs programmes utilisateurs et les transmet à l'entité suivante. Pour ce faire, elle ajoute des informations complémentaires à chaque paquet y compris des codes indentifiant le programme d'application émetteur et récepteur ainsi qu'une somme de contrôle (ou *checksum*). L'entité TCP réceptrice utilise cette somme de contrôle pour vérifier si le paquet reçu est intact, et utilise le code de destination pour identifier le programme d'application auquel le paquet devrait être livré.

Le modèle TCP/IP recherchait, avec le protocole IP, l'interconnexion des réseaux, il va essayer, avec le protocole TCP, de satisfaire les exigences de résistance aux incidents et de disponibilité en cas d'encombrement. Ce protocole TCP offre un service de communication fiable entre processus appartenant à des ordinateurs reliés à des sous-réseaux éventuellement distincts. Ce service est de type circuit virtuel, c'est-à-dire orienté connexion.

4. la couche application:

Au niveau le plus élevé, les utilisateurs invoquent des programmes d'application qui accèdent à l'Internet. Une application interagit avec le ou les protocoles de niveau transport pour transmettre ou recevoir des données. La couche application peut également accèder directement à IP. Chaque programme d'application choisit sa propre forme de données, qui peut être une séquence de messages ou un flot de bytes. Quelque soit la forme, la couche application passe les données à la couche transport pour la livraison;
Le schéma ci-dessous illustre la structure générale de TCP/IP:



Les protocoles ARP/RARP, IP, ICMP, UDP et TCP seront détaillés dans les chapitres suivants: respectivement: II.2., II.3, II.4., II.5., II.6.

D'autres protocoles utilisent également le protocole IP pour transporter des informations relatives à la gestion de l'Internet. Il s'agit des protocoles GGP (*Gateway to Gateway Protocol*), RIP (*Routing Information Protocol*), EGP (*Exterior Gateway Protocol*). Tous ces protocoles aident les passerelles dans leur fonction de routage des datagrammes, ils ne seront pas développés dans ce texte.

Chaque ordinateur hôte qui veut utiliser un réseau interconnecté basé sur le modèle TCP/IP doit disposer d'un module IP et d'un module TCP. L'unité de données échangée entre deux modules TCP s'appelle le "segment TCP".

Pour pouvoir offrir une communication fiable entre deux ordinateurs, le protocole TCP utilise les mécanismes suivants:

- L'acquittement positif et la retransmission.
- Le contrôle de flux.
- Le séquencement.
- Le multiplexage.
- Les connexions.
- Les priorités et la sécurité.

Tous ces mécanismes sont généralement connus sauf peut être la technique de multiplexage. Celle-ci est totalement différente de celle utilisée dans la terminologie OSI. Le multiplexage TCP/IP permet à plusieurs processus d'un même ordinateur de communiquer simultanément par le même module TCP. Pour y parvenir, le protocole TCP fournit un ensemble de portes dans chaque ordinateur. Ces portes permettent au module TCP d'identifier les processus et d'associer les données, reçues ou à envoyer, à un processus. Un numéro de porte (ou *port*) concaténé à l'adresse internet forme un *socket*. Une paire de *sockets* identifie une connexion.

La plupart des systèmes d'exploitation fournissent des accès synchrones aux ports. Les paquets reçus sont mis dans une file d'attente particulière d'un port. Ces paquets seront traités ultérieurement. De même, le système d'exploitation bloque les processus qui tentent d'extraire les données d'un port avant qu'un message arrive.

Pour communiquer avec un autre port, un émetteur doit connaître l'adresse Internet de la machine de destination et le numéro de port de destination du processus. Avec chaque message, l'émetteur fournit un numéro de port de la machine de destination à laquelle le message est destiné ainsi qu'un numéro de port de la machine source à laquelle les réponses doivent être adressées. Ainsi, il est possible pour chaque processus recevant un message de répondre à son émetteur.

Deux machines doivent s'accorder sur les numéros de ports avant de commencer les échanges. Par exemple, quand une machine A veut obtenir un fichier d'une machine B, elle a besoin de connaître le numéro de port utilisé par le programme de transfert de fichiers sur la machine B. Il y a deux approches fondamentales pour l'affectation des ports:

La première approche consiste à donner à une autorité centrale le droit d'assigner les numéros de ports nécessaires et de publier la liste de ces numéros. Les logiciels seront ainsi construits en respectant cette liste. Cette approche est appelée affectation universelle et les numéros de port assignés spécifiés par l'autorité sont appelés les affectations de port connues.

La deuxième approche de l'affectation des ports utilise des liaisons dynamiques. Dans cette approche les ports ne sont pas connus globalement. Chaque programme qui a besoin d'un port se voit affecté un numéro de port par le logiciel du réseau. Pour connaître l'affectation du port courant sur une autre machine, il est nécessaire d'envoyer une demande de la forme: comment atteindre le service de transfert de fichiers. La machine cible répond en donnant le numéro de port correct qui doit être utilisé.

Les concepteurs Internet ont adoptés une approche hybride qui affecte à quelques numéros de ports une priorité, ceux-ci sont disponibles pour des sites locaux ou pour les programmes d'application. Les numéros de ports affectés ont leurs 8 bits d'ordre bas mis à zéro. Ainsi, seulement 255 ports peuvent être affectés. Ces numéros de ports sont les mêmes pour TCP et UDP.

Dans le modèle TCP/IP, on a aussi voulu que certains protocoles d'application puissent encore disposer d'un service de type datagramme au niveau correspondant à la couche de transport. Ceci explique la présence du protocole UDP qui, par rapport au protocole IP, n'ajoute que la fonction de multiplexage.

Les protocoles TCP et UDP ne présentent aucun intérêt s'ils ne sont pas utilisés par des logiciels d'application. Le modèle TCP/IP a choisi d'associer à chaque application un protocole particulier qui utilisera un service de transport fourni, soit par TCP, soit par UDP. Ainsi, pour l'application de transfert de fichiers, on retrouve, au niveau correspondant à la couche application, les protocoles FTP et TFTP. Le premier utilise le protocole TCP tandis que le second utilise le protocole UDP. Cette couche comprend également un protocole de terminal virtuel, Telnet qui utilise TCP, un protocole de messagerie, SMTP (*Simple Mail Transfer Protocol*) qui utilise TCP, un protocole de serveur de nom, NSP (*Name Server Protocol*) qui utilise UDP et bien d'autres protocoles.

Dans ce chapitre, nous venons de dégager les quelques idées maîtresses qui ont régi la conception du modèle TCP/IP. Ce modèle, composé de quatre couches, s'articule autour des protocoles IP et TCP. Le protocole IP vise essentiellement l'interconnexion des réseaux tandis que le protocole TCP cherche à offrir un service de communication fiable. Dans ce qui suit, nous allons présenter la démarche suivie par l'ISO ainsi qu'une comparaison des deux modèles.

II.1.3. TCP/IP face à OSI

II.1.3.1. L'architecture en couches du modèle OSI

Tout comme le modèle TCP/IP, le modèle OSI essaie d'apporter une solution aux problèmes que pose l'interconnexion de réseaux. Le problème principale est l'interconnexion des réseaux ayant des protocoles, des services, des interfaces, et des formats de paquets

différents. D'autres problèmes sont reliés à l'organisation des sessions, à la protection ou à la référence des fichiers, etc.

Pour remédier à ces problèmes, TCP/IP et OSI proposent des architectures hiérarchiques qui ne résultent des efforts de plusieurs constructeurs.

John D. Day et Hubert Zimermann expliquent dans l'article de référence qu'ils ont consacré au modèle OSI [Zimmermann 83] la démarche suivie par l'ISO lors de l'élaboration de ce modèle. Ils y déclarent ceci: "Dans OSI, le problème a été abordé de manière *top-down*, en partant d'une description d'un haut niveau d'abstraction qui impose peu de contraintes, et procédant à des descriptions de plus en plus raffinées, elles intègrent des contraintes de plus en plus strictes. Dans le monde OSI, on reconnaît trois niveau d'abstraction: l'architecture OSI, les spécifications de service et les spécifications de protocole".

L'architecture OSI est spécifiée très scrupuleusement par la norme ISO 7498 et ses annexes. Elles se composent de deux parties. La première décrit les éléments de cette architecture tandis que la seconde énumère les services et les fonctions des sept couches choisies par l'ISO. L'idée de base de la division en couches est que chaque couche ajoute de la valeur aux services fournis par l'ensemble des couches inférieurs de telle sorte que la couche la plus haute dispose de l'ensemble complet des services nécessaires pour exécuter les applications. L'objectif de cette division en couches a été déjà expliqué dans le paragraphe II.1.2.2. de ce texte.

Le deuxième aspect est occupé par des spécifications des services OSI. Ce niveau permet de mettre en évidence la distinction qui existe entre utilisateurs et fournisseurs de services.

Le troisième aspect est celui des spécifications des protocoles OSI. Chaque spécification de protocole définit très précisément quelle information de contrôle doit être envoyée et quelles procédures doivent être utilisées pour interpréter cette information de contrôle [Zimmermann 83].
Cette architecture en 7 couches est illustrée dans la figure ci-dessous:

Le modèle à 7 couches de l'ISO:

Couches	Fonctionnalités
7	Application
6	Présentation
5	Session
4	Transport
3	Réseau
2	Liaison de donnée
1	Physique

Plusieurs protocoles sont associés au modèle OSI. Ils sont ensuite adaptés par les réseaux. Ces réseaux se composent de commutateurs de paquets complexes qui contiennent l'intelligence nécessaire pour le routage des paquets. Les hôtes ne sont pas directement connectés au réseau de communication. Chacun des hôtes est relié à un commutateur de paquets utilisant une ligne de communication. La connexion entre un hôte et un commutateur de paquets est généralement un réseau minuscule composé d'une liaison en série. Le hôte doit suivre une procédure complexe pour transférer des paquets sur le réseau.

Chacune des sept couches a les spécifications suivantes:

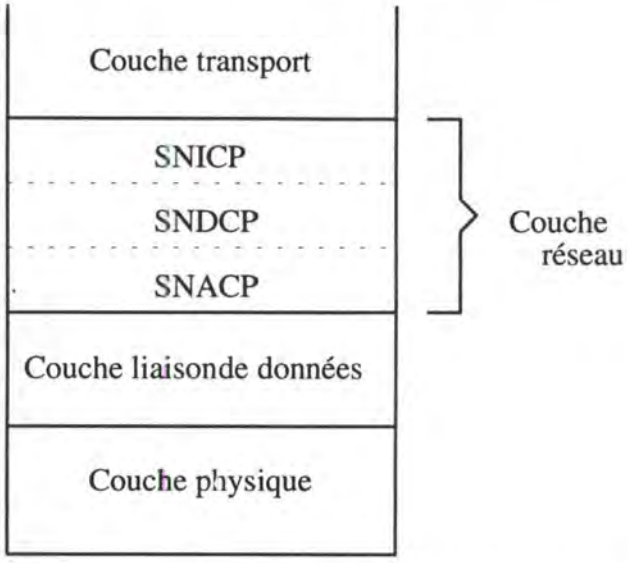
- **Couche physique:** Cette couche spécifie un standard pour l'interconnexion physique entre les ordinateurs hôtes et les commutateurs de paquets, elle spécifie également les procédures utilisées pour le transfert des paquets d'une machine à une autre. Dans le modèle de référence, le niveau 1 spécifie l'interconnexion physique ainsi que les caractéristiques du voltage et du courant électriques.

- **Couche de liaison de données:** Le niveau 2 spécifie la manière dont la donnée est transmise entre un hôte et un commutateur de paquets. On utilise généralement le terme "trame" pour désigner une unité de donnée ainsi transmise. Et comme le matériel livre seulement un flot de bits, le protocole de niveau 2 doit définir le format des trames et doit spécifier la manière dont les deux machines reconnaissent les limites de ces trames. Ce protocole de niveau 2 dispose également d'un système de détection d'erreurs car les erreurs de transmission peuvent détruire des données. La transmission étant non fiable, l'échange d'accusés de réception permet aux deux machines de savoir si une trame est bien reçue.

- **Couche réseau:** Le modèle ISO spécifie des fonctionnalités qui viennent compléter la définition de l'interaction entre un hôte et un réseau. Cette couche définit l'unité de transfert de base qui traverse le réseau, ainsi que les concepts d'adressage de destination et de routage. Un réseau pourrait permettre aux paquets définis par les protocoles de niveau 3 d'être plus large que la taille des trames qui peuvent être transmis au niveau 2. Le logiciel du niveau 3 assemble un paquet dans la forme exigée par le réseau et utilise le niveau 2 pour l'envoyer, éventuellement en pièces, au commutateur de paquets. Le niveau 3 doit également régler les problèmes d'encombrements du réseau.

  D'une façon plus générale, la structure proposée par l'ISO pour cette couche, représentée dans la figure ci-dessous, se compose de trois sous-couches. Au niveau le plus bas, le protocole SNACP (*SubNetwork ACces Protocol*) assure les fonctions de réseau qui sont spécifiques à un sous-réseau réel particulier, et qui concernent en particulier le routage dans le sous réseau. Cette sous-couche ne fournit pas toujours le service désiré. Pour faire face à cette situation et pour adapter ce protocole SNACP, deux sous-couches supplémentaires ont été ajoutées. Au niveau le plus élevé, la fonction d'interconnexion est prise en charge par un protocole SNICP (*SubNetwork Independent Convergence Protocol*) qui fournit à la couche de transport un service de réseau OSI, et qui exploite un jeu bien défini de fonctions offertes par un protocole intermédiaire appelé SNDCP (*SubNetwork Dependent Convergence Protocol*). Le protocole SNICP est donc indépendant des caractéristiques du sous-réseau réel. le protocole SNDCP sert essentiellement à assurer un découplage entre les caractéristiques qui sont liées au sous-réseau et l'organisation du protocole qui fournit le service de réseau à la couche transport.

Organisation interne de la couche réseau



- **Couche de transport:** Le niveau 4 fournit un transfert de données fiable de bout en bout entre un le hôte de destination et le hôte source. Cette fiabilité est différente de celle offerte par les basses couches des protocoles. Celles-ci offrent des vérifications de fiabilité au niveau réseau à chaque transfert.

- **Couche session:** Cette couche s'occupe de la gestion des données qu'elle reçoit de la couche du dessus ou du dessous. Elle fournit des services de transfert, de contrôle et de gestion des données sur une connexion de sessions. Ceci améliore le service fiable de transfert de données de bout en bout offert par la couche de transport d'au-dessous.

- **Couche présentation:** Cette couche s'occupe de la représentation des données. Elle fournit une représentation commune pour toutes les informations des applications quand elles transitent entre les processus d'application paires [Henshall 90]. La couche se compose de fonctions nécessaires aux programmes d'application utilisant le réseau. Les routines standards qui permettent de compresser les textes ou de convertir les images graphiques en séquences de bits pour leur transmission sur le réseau est un exemple typique des fonctions de cette couche.

- **Couche application:** Finalement, cette couche s'occupe des programmes application qui utilisent le réseau. Le courrier électronique ou les programmes de transfert de fichiers sont les exemples les plus connus.

II.1.3.2. Comparaison des deux modèles

Au niveau de leurs architectures, la comparaison va s'articuler autour des quatre couches du modèle TCP/IP. Avant de passer en revue chacune de ces couches, nous rappelons au tableau ci-dessous la correspondance généralement admise entre les couches des deux modèles [Stalling 85].

Comparaison des architectures OSI et TCP/IP:

<table>
<tr><th>Modèle OSI</th><th>Modèle TCP/IP</th></tr>
<tr><td>application</td><td rowspan="3">applications</td></tr>
<tr><td>présentation</td></tr>
<tr><td>session</td></tr>
<tr><td>transport</td><td>bout en bout (TCP)</td></tr>
<tr><td>SNICP
SNDCP
SNACP</td><td>inter-réseaux (IP)</td></tr>
<tr><td>liaison</td><td rowspan="2">accès-réseau</td></tr>
<tr><td>physique</td></tr>
</table>

"Le but poursuivi par OSI est de permettre à n'importe quel ordinateur situé n'importe où dans le monde de communiquer avec un autre ordinateur tant que ceux-ci respectent les normes OSI" [Zimmermann 83]. Ceci entraîne de la part de l'ISO, du CCITT et de l'ECMA une grande activité dans la définition de normes dans les couches 1,2 et 3. A l'opposé, le modèle TCP/IP peut utiliser presque tous les réseaux disponibles sur le marché en respectant les objectifs définis par le DoD. Ceci explique l'intérêt relatif manifesté par le DoD au niveau accès réseau. A ce niveau, nous pouvons qualifier la philosophie OSI de restrictive dans la mesure où les systèmes doivent respecter les normes OSI tandis que la philosophie DoD est résolument libérale car le modèle DoD vise à l'interconnexion de tout système commercialisé.

Les principaux services offerts par la couche inter-réseaux du modèle DoD et la couche réseau du modèle OSI varient d'un modèle à l'autre. Le niveau inter-réseau offre avec le protocole IP un service datagramme unique tandis que la couche réseau d'OSI fournit un service datagramme (sans connexion) analogue à celui d'IP et un service circuit virtuel (avec connexion). Le service IP du modèle DoD a opté pour un service datagramme. Ce service permet:

1. l'utilisation des passerelles pour éviter la gestion des informations d'états concernant chaque circuit crée,
2. continuer les communications de bout en bout par des chemins alternatifs dans le cas où une passerelle ou un réseau viendrait de tomber en panne,
3. et laisser une très grande marge de manoeuvre pour définir de nouveaux protocoles autres que TCP et UDP au niveau bout en bout.

En ce qui concerne la fonction d'interconnexion des réseaux, le modèle DoD propose une solution unique: l'interconnexion des réseaux au moyen de passerelles implémentant le protocole IP. Le modèle OSI propose, quant à lui, de nombreuses solutions, mais celles-ci ne sont pas encore toutes complètement spécifiées. Le solution proposée par l'ISO est fort analogue à celle du protocole IP, c'est pourquoi il est également appelé protocole CL-IP (*ConnectionLess - Internet Protocol*) ou ISO-IP (*International Standards Organization - Internet Protocol*). Ce protocole est typiquement un protocole SNICP. On peut également remarquer que le protocole IP du DoD recouvre les fonctions des protocoles SNDCP et

SNICP dans la mesure où il doit également harmoniser les services offerts par les différents réseaux.

Le niveau bout en bout du DoD ainsi que la couche transport de l'ISO offrent deux types de service transport: un service transport orienté "connexion" et un autre orienté "sans connexion". Ce dernier est prévu pour les applications temps réel portant sur l'acquisition et la distribution de données. Le service orienté "connexion" est bien adapté à la plupart des autres applications informatiques; il est le plus utilisé.

Pour assurer ces deux services, le modèle DoD a spécifié deux protocoles: le protocole TCP pour offrir le service circuit virtuel et le protocole UDP pour le service datagramme. Cette simplicité du modèle DoD repose sur le fait que le niveau inter-réseau offre un service unique de transport non fiable

Le modèle OSI, pour sa part, en a défini quatre pour offrir un service de transport orienté "connexion" et un pour le service sans connexion. Dans ce modèle, la couche transport est confrontée à une diversité de services offerts par la couche réseau.

II.1.3.3. Les principales différences entre les deux modèles

Il y a deux différences importantes entre le protocole des couches Internet et OSI. La première différence tourne autour de la fiabilité, alors que la deuxième concerne la localisation de l'intelligence dans tout le système.

II. 1.3.3.1. Bout en Bout contre fiabilité niveau liaison

Une différence majeure entre les protocoles TCP/IP et les protocoles OSI est liée au contrôle d'erreurs. Dans le modèle OSI, ce contrôle se fait entre les différents niveaux. Cependant, dans le modèle TCP/IP le contrôle se fait de bout en bout.

En général, dans le modèle OSI, le logiciel du protocole détecte et manipule les erreurs à tous les niveaux. Au niveau liaison de données, des protocoles complexes garantissent un transfert correct entre un hôte et le commutateur de paquets avec lequel il est connecté. Les sommes de contrôle (*checksum*) accompagnent chaque paquet de donnée transféré, et le récepteur accuse réception de chaque paquet reçu. Le protocole de niveau liaison comprend un délai d'attente (*timeout*) et des algorithmes de retransmission qui empêchent la perte de données et fournit un redressement automatique après une panne éventuelle du matériel. Tous les niveaux fournissent, ainsi, une fiabilité de la transmission entre les machines.

Au niveau 3, par exemple, du modèle OSI un système de détection d'erreurs et de récupération des paquets transférés sur le réseau est offert, il utilise les sommes de contrôle ainsi que les techniques du délai d'attente et de retransmission. Au niveau 4, un service de fiabilité de bout en bout est également offert, il a une correspondance entre la source et la destination pour vérifier la livraison.

Internet fonde ses couches de protocoles sur l'idée que la fiabilité est un problème de bout en bout. La philosophie de l'architecture est simple: construire l'Internet de façon à ce qu'il peut manipuler le chargement prévu, mais autorise des liaisons ou des machines individuelles de perdre des données ou de les corrompre sans essayer de les récupérer de façon systématique.

En fait, il n'y a aucune couche de liaison qui s'occupe de la fiabilité dans la plupart des logiciels Internet. C'est plutôt la couche transport qui est chargée de la détection des erreurs et du problème de récupération. Le niveau liaison, libéré de ces vérifications, rend le logiciel d'Internet plus organisé et facile à implémenter. Les passerelles intermédiaires peuvent écarter des datagrammes corrompus dus aux erreurs de transmission, ils peuvent écarter tous les datagrammes qui ne peuvent pas être livrés. Ils peuvent également écarter les datagrammes quand le taux d'arrivées dépasse la capacité de la machine, en redirigeant ces datagrammes vers d'autres chemins pour les retarder et ceci sans informer la source ni la destination. Les liaisons non fiables ont pour conséquence que quelques datagrammes peuvent ne pas arriver à destination. La détection et la récupération des datagrammes perdus est réalisé entre la source et la dernière destination et pour cela elle est appelée une vérification de bout en bout. Le logiciel qui s'occupe de cette fiabilité de bout en bout se trouve dans la couche transport, il utilise les sommes de contrôle, les accusés de réception, et les délais d'attente pour contrôler la transmission. Ainsi à la différence du modèle ISO, le logiciel du protocole Internet concentre le contrôle de fiabilité dans une seule couche.

II.1.3.3.2. Le contrôle

Une autre différence entre le modèle OSI et le modèle Internet apparaît quand on veut situer le lieu exact de l'autorité et du contrôle.

En règle général, les principaux réseaux du modèle OSI sont considérés comme étant des utilitaires qui fournissent un service de transport. Les vendeurs qui offrent le service contrôlent l'accès au réseau et le trafic des moniteurs pour l'enregistrement dans la comptabilité et pour la facturation. Le vendeur de réseaux s'occupe des problèmes tels que: le routage, le contrôle de flux, les accusés de réception internes et la fiabilité des transferts. Le réseau est un système complexe et indépendant par lequel on peut atteindre des ordinateurs hôtes; les hôtes participent eux-mêmes très peu dans les opérations des réseaux. De plus, si l'on considère l'interconnexion dans le modèle OSI, on est très proche de ce qui se fait dans TCP/IP.

l'Internet, cependant, exige des hôtes de participer dans presque tous les protocoles du réseau. Comme c'est déjà mentionné ci-dessus, les hôtes implémentent de façon active les détections d'erreurs de bout en bout et leurs corrections. Ils participent également dans le routage puisqu'ils doivent choisir une passerelle quand ils envoient des datagrammes, ils participent aussi dans le contrôle du réseau puisqu'ils doivent manipuler les messages de contrôles ICMP. Ainsi, le réseau Internet peut être vu comme un système de livraison de paquets relativement simple lié à des hôtes intelligents [Comer 88].

II.1.3.4. Migration vers le standard OSI

Etant donné que la migration vers le standard OSI est un but avoué (si pas réel) pour tout le monde, il paraît aussi évident que cette migration ne se fera pas du jour au lendemain sans problème. Il faudra donc dans un premier temps composer avec les solutions existante de la meilleure manière possible. C'est-à-dire qu'il faudra composer avec le standard de fait qu'est TCP/IP avant de pouvoir migrer totalement vers le standard OSI issu d'un travail sérieux de modélisation.

Dans la littérature, quand on parle de stratégies de migration vers OSI, on parle aussi très souvent comme solution existante de TCP/IP qui sont les protocoles de communication les plus répandus actuellement dans le monde. Les principales raisons avancées par certains spécialistes pour s'intéresser à la solution TCP/IP dans une première étape de la migration vers le standard OSI, sont expliquées ci-dessous.

Le standard OSI a rencontré et continue de rencontrer certaines réticences de la part des utilisateurs et ceci est entre autre dû au fait qu'il existe une solution bien établie, qui a déjà pu être rôdée et qui maintenant a fait preuve de fiabilité. C'est la solution TCP/IP. Les utilisateurs se méfient des solutions nouvelles et préfèrent partir sur de bonnes bases en faisant bon emploi de l'expérience acquise. Ceci dit, on peut sans trop de risques faire confiance à la fiabilité des protocoles des couches basses de la pile OSI. Mais on sait que le modèle OSI propose souvent plusieurs variantes de protocoles et qu'il n'y a pas encore accord sur le choix de ces protocoles. On pense ici à la dissension entre les modes connectés et les modes non connectés par exemple. Il se fait que les standards OSI pour le mode non connecté n'ont pas encore atteint un degré satisfaisant de maturité. Voilà donc une première raison de se tourner vers la solution TCP/IP, du moins temporairement, pour ceux qui ont plus confiance dans le mode non connecté. Il s'agit de la confiance en une solution qui est le fruit d'une dizaine d'années d'expérience et qui, de plus, est répandue; face à la méfiance en une solution nouvelle dont l'attrait est surtout la cohérence théorique (grâce à un découpage logique en couches) mais qui n'a pas encore pu faire ses preuves sur le plan pratique.

Le but de tous les architectes de réseaux est de trouver une architecture telle que le maximum d'hôtes (et donc de personnes) puissent y être connectés. Ils veulent aussi pouvoir utiliser le maximum de produits proposés sur le marché. C'est bien cela que la nécessité d'un standard ne se discute plus mais c'est aussi pour cette raison qu'un architecte de réseau doit faire des choix stratégiques pour que son réseau rencontre autant que possible ces exigences tout de suite.

Dans le même ordre d'idée, on dit qu' "une des faiblesses de OSI, ce sont ses standards et le temps qu'il faut pour les établir". Quand on a des décisions à prendre pour un réseau, on ne peut pas toujours attendre que le standard ait été approuvé par toute la communauté et ait été de plus implémenté par toute la communauté. Une deuxième raison de se tourner vers la solution TCP/IP vient donc du défaut de lenteur de développement des produits OSI et surtout la lourdeur de ses différents protocoles.

Chapitre II.2.

L'ADRESSAGE INTERNET ET LES PROTOCOLES ARP ET RARP

II.2.1. L'adressage Internet

II.2.1.1. Introduction

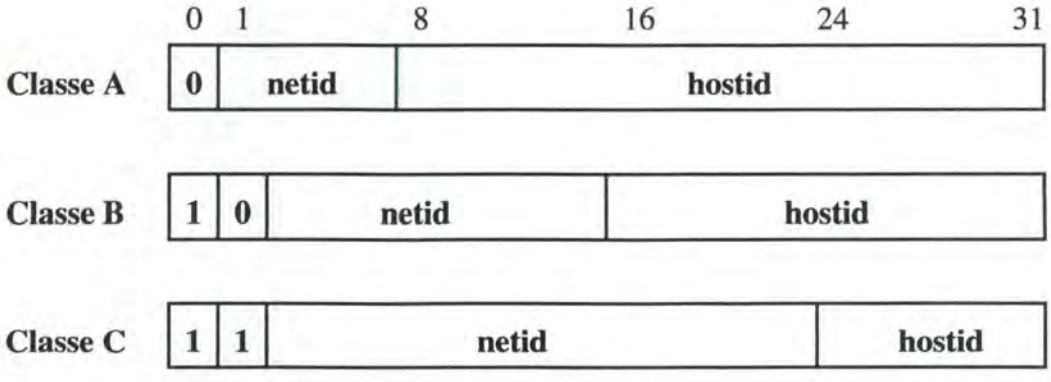
Internet est un grand réseau virtuel construit par l'interconnexion des sous-réseaux physiques avec des passerelles. Il est conçu et implémenté entièrement sous forme d'un logiciel. Les concepteurs ont eu donc toute la liberté de choisir les formats et les tailles des paquets, les adresses, les techniques de livraison, etc. Rien n'a été imposé par le matériel. Pour les adresses, ils ont choisi un plan analogue à celui des réseaux physiques dans lesquels on attribue à chaque hôte une adresse sous forme d'un entier. Cet entier s'appelle l'adresse Internet.

A chaque hôte sur le réseau Internet est affecté une adresse Internet unique d'une longueur de 32 bits qui sera utilisé dans toutes les communications. Cet identificateur est universel. Les bits d'une adresse Internet de tous les hôtes sur un sous-réseau donné ont un préfixe commun.

II.2.1.2. Les adresses Internet

Chaque adresse est représenté par une paire (*netid, hostid*), où "*netid*" est l'identifiant du réseau, et "*hostid*" l'identifiant du hôte sur ce réseau. En pratique, les adresses Internet se présentent principalement sous trois formes, comme l'illustre la figure ci-dessous:

les trois formes principales des adresses internet:



Les adresses de la classe A sont utilisées pour des réseaux qui ont plus de 2^{16} (65636) hôtes, 7 bits sont consacrés au "*netid*" et 24 au "*hostid*". Cependant, les adresses de la classe B qui

sont utilisées pour des réseaux de taille moyenne c'est-à-dire entre 2^8 (256) et 2^{16} hôtes, allouent 14 bits au "*netid*" et 16 bits au "*hostid*". Finalement, les adresses de la classe C qui ont moins de 2^8 hôtes, allouent 22 bits au "*netid*" et seulement 8 bits au "*hostid*".

Cette structure des adresses Internet permet l'extraction des zones "*netid*" et "*hostid*" en un temps constant . Les passerelles, qui sont à la base du routage à partir de "*netid*", dépendent de cette extraction efficace.

Une adresse Internet ne peut pas toujours être le seul identifiant d'un hôte, car les passerelles qui relient deux réseaux physiques ont nécessairement deux adresses Internet. La plupart des machines ont plus d'une connexion physique. Ces machines exigent plusieurs adresses Internet; comme ces adresses spécifient un réseau et un hôte sur ce réseau, ils n'identifient pas un hôte mais une connexion à un réseau. Ainsi, une passerelle qui connecte n réseaux possède n adresses Internet différentes, une pour chaque connexion réseau.

Le codage des informations du réseau en adresses Internet n'est pas le seul moyen de rendre le routage efficace, ces adresses Internet peuvent également spécifier des réseaux.

Une adresse Internet dont tous les bits de son "*hostid*" mis à zéro est utilisé pour faire référence à un réseau. Un hôte ayant ce type d'adresse ferait référence à lui même. Et si le champ "*netid*" d'une adresse Internet est constitué de zéros, le hôte ferait référence à ce même réseau.

Un autre avantage de cet adressage est qu'il comprend une adresse de diffusion qui fait référence à tous les hôtes sur le réseau. D'après les standards, si les bits du champ "*hostid*" d'une adresse sont mis à 1, celle-ci est réservée à la diffusion. Sur Ethernet (cf. I.1.) la diffusion peut être aussi efficace que la transmission normale, sur d'autre technologies elle est assurée par les modules du réseau et est plus lente qu'une simple transmission. La diffusion peut être absente sur certains réseaux.

Quand une machine reçoit un paquet avec le champ "*netid*" mis à zéro et le champ "*hostid*" égale à l'identifiant du hôte, ce paquet est interprété comme étant transmis du même réseau sur lequel les deux machines sont attachés et à travers lequel le paquet est arrivé.

On utilise cette technique dans le cas où on ne connaît pas l'adresse réseau avec lequel on veut communiquer. Le hôte utilise le "*netid*" zéro temporairement pour pouvoir recevoir des réponses des autres hôtes avec l'adresse complète du réseau et ainsi il pourrait l'enregistrer et l'utiliser pour une prochaine émission.

Les adresses Internet se composent de quatre entiers séparés par des points, où chaque entier donne la valeur d'un octet de l'adresse Internet. Pour les réseaux de la classe A, le premier entier varie de 0 à 128, pour la classe B, il se situe entre 128 et 192, et pour la classe C, il est de 192 à 256.

Tous les adresses Internet sont assignées par une autorité centrale qui s'appelle le NIC (*Network Information Center*). Le NIC attribue des numéros à la partie réseau de l'adresse Internet et délègue la responsabilité de l'attribution de numéros de hôtes à l'organisme demandeur.

Il attribue aux réseaux locaux, tel que Ethernet, les numéros de la classe C car la plupart de ces réseaux ne dépassent pas les 255 hôtes. Aux grands réseaux, tel que ARPANET, sont attribué les numéros de la classe A.

Le NIC s'occupe seulement de l'attribution d'adresses Internet aux réseaux reliés ou qui seront reliés à l'Internet DARPA. Plusieurs autres groupes qui utilisent les protocoles TCP/IP attribuent eux-mêmes les adresses réseaux car ils n'ont pas besoin de s'interconnecter avec Internet.

II.2.1.3. Faiblesse de l'adressage Internet

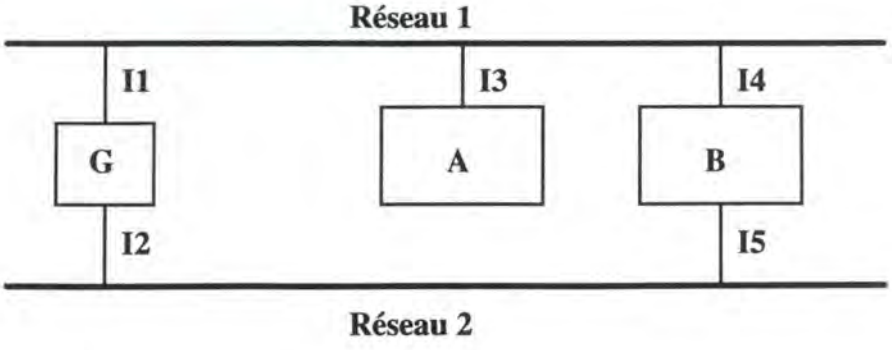
Le codage des adresses Internet a quelques désavantages. Le plus évident est que les adresses dépondent des connexions et non des hôtes. Si on déplace un hôte d'un réseau à un autre, son adresse Internet devrait changer.

Une autre faiblesse de ce système d'adressage est que si un réseau de classe C dépasse 255 hôtes, les adresses Internet de ces derniers doivent être changé au système d'adressage de la classe B. Ce problème paraît assez simple, mais en réalité il est difficile de corriger les adresses d'un réseau et cela prend beaucoup de temps. Et comme la majorité des logiciels ne peuvent pas manipuler plusieurs types d'adresses pour le même réseau physique, les administrateurs ne peuvent agir de façon progressive pour introduire les nouvelles adresses. Ils doivent arrêter complètement l'utilisation des anciennes adresses du réseau et commencer l'utilisation des nouvelles dès que la correction est faite.

Comme on l'a déjà indiqué, le routage sera basé sur ces adresses Internet et particulièrement sur l'identifiant du réseau pour pouvoir prendre des décisions de routage. Si on considère un hôte ayant deux connexions au réseau Internet. On sait qu'un tel hôte dispose de plus d'une adresse Internet. Ainsi, le chemin pris par les paquets pendant la transmission dépend de l'adresse utilisée.

Connaissant une seule adresse d'un hôte de destination, il peut être parfois insuffisant car il pourrait être impossible de l'atteindre en utilisant cette adresse.
Considérons l'exemple suivant illustré ci-dessous:

Faiblesse de l'adressage Internet:



Les deux hôtes A et B peuvent communiquer directement en utilisant le réseau 1. Ainsi, A s'adresse à B en utilisant l'adresse Internet I4. Un second chemin existe entre A et B à travers

la passerelle G et est utilisé quand A envoie des paquets à l'adresse Internet I5. Si la connexion de B sur le réseau 1 échoue, et que la machine reste active, A ne pourrait plus atteindre B avec l'adresse I4. Pour pouvoir l'atteindre il faudrait spécifier l'adresse I5. Donc, si l'interface I4 se déconnecte, A devrait utiliser I5 pour atteindre B, en routant les paquets à travers la passerelle G. Ainsi, Chaque hôte devrait connaître toutes les adresses des autres car en cas de panne d'une connexion il peut saisir une autre adresse encore disponible.

II.2.2. La résolution du problème des adresses (ARP)

II.2.2.1. Objectifs

Considérons deux machines A et B sur un même réseau. Chacune d'entre elles possède une adresse Internet (respectivement I_A et I_B) et une adresse physique (respectivement P_A et P_B). L'objectif est de combiner les logiciels de bas niveau qui cachent les adresses physiques et de permettre aux programmes de haut niveau de travailler avec seulement des adresses Internet. La communication finale doit être, cependant, réalisé par des réseaux physiques utilisant n'importe quel type d'adresse physique fournit par le matériel. La question qui se pose est: si une machine A veut envoyer un paquet à une autre machine B sur le réseau qui les relie, et qu'elle ne dispose que de l'adresse Internet I_B de B, comment peut elle transformer cette adresse en une adresse physique P_B ?

Le problème de la transformation des adresses Internet en des adresses physiques est connu sous le nom du problème de résolution des adresses. Ce problème a été résolu de plusieurs façons. Quelques protocoles Internet gardent des tables dans chaque machine, ces tables contiennent des paires d'adresses Internet-physiques. D'autres ont résolu le problème en encodant les adresses physiques en des adresses Internet. En utilisant la méthode des tables nous pouvons rendre l'adressage Internet difficile.

En fait, à chaque hôte est assignée une adresse de 32 bits. Cette adresse est appelée l'adresse Internet. Ces adresses sont utilisées pour envoyer ou pour recevoir les paquets. Mais pour que deux machines puissent communiquer physiquement sur un réseau, ils ne peuvent utiliser que des adresses physique. Pour cela, l'adresse internet doit être convertie en une adresse physique. Le protocole ARP (pour *Address Resolution Protocol*) nous permet de résoudre ce problème.

II.2.2.2. La structure statique du protocole ARP

II.2.2.2.1. L'encapsulation du message ARP

Quand les messages ARP se déplacent d'une machine à une autre, ils doivent être transportés dans des trames physiques. Ils sont ainsi encapsulés dans des trames Ethernet.

Encapsulation d'un message ARP dans une trame Ethernet:

L'en-tête de la trame	Le message ARP complet traité comme une donnée

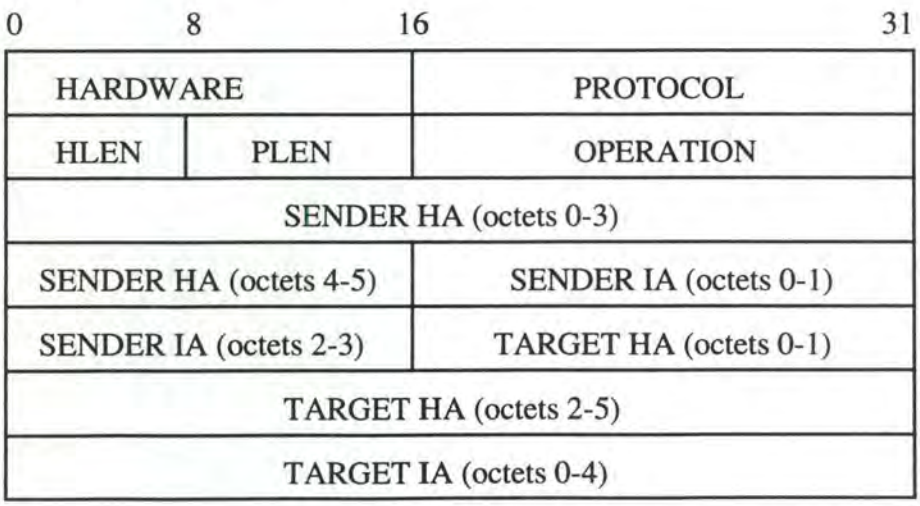
Pour voir si la trame transporte une requête ou une réponse ARP, l'émetteur affecte une valeur spéciale dans le champ TYPE de l'en-tête de la trame et place le message ARP dans le champ de donnée de la trame. Quand une trame arrive à un hôte, le système examine son type pour déterminer son contenu. Par exemple, sur Ethernet, les requêtes ARP ont un champ type d'une valeur 0806_{16} (base 16) et les réponses ont un type d'une valeur de 8035_{16}. Ceux-ci sont des valeurs standards, affectées par l'autorité qui a établi les standards Ethernet.

II.2.2.2.2. Le format du protocole ARP

Le format du message ARP est différent de celui des autres protocoles. Les données dans les paquets n'ont pas un en-tête de format fixe. Le message est conçu pour être utilisé par une variété de technologie de réseau, ainsi les premiers champs de l'en-tête contiennent des comptes qui spécifient les longueurs des champs suivants. En fait, ARP peut être utilisé avec des adresses physiques et des adresses de protocoles arbitraires.

L'exemple de la figure ci-dessous montre le format du message ARP d'une longueur de 28 octets utilisé sur le support Ethernet (où les adresses physiques sont de 48 bits de longueur), quand on résout les adresses du protocoles Internet du DARPA (d'une longueur de 4 octets). A la différence de la majorité des protocoles Internet, les champs de longueurs variables dans les paquets ARP ne s'alignent pas sur les limites de 32 bits, rendant ainsi le diagramme difficile à lire. Par exemple, l'adresse de la machine de l'émetteur SENDER HA, occupe 6 octets successifs, alors elle prend deux lignes dans le diagramme. Néanmoins, ce format a été choisi parce qu'il est un standard dans la littérature d'Internet.

Le format des messages ARP/RARP utilisé pour la résolution d'adresses Internet --> Ethernet:



Le champ HARDWARE spécifie le type de l'interface de hardware pour laquelle l'émetteur cherche une réponse; elle a une valeur de 1 pour le support Internet. Le champ OPERATION spécifie une requête ARP (1), ou une réponse ARP (2), ou une requête RARP (3), ou encore une réponse RARP (4). Les champs HLEN et PLEN permettent au protocole ARP d'être utilisé avec des réseaux arbitraires puisqu'ils spécifient la longueur de l'adresse physique de la machine et la longueur de l'adresse du protocole. L'émetteur fournit l'adresse de sa machine

ainsi que son adresse Internet, s'ils sont connus, dans les champs SENDER HA et SENDER IA.

Quand il envoie une requête, l'émetteur fournit également l'adresse Internet de destination (ARP), ou l'adresse de la machine de destination (RARP) en utilisant les champs TARGET HA et TARGET IA. Une réponse transporte à la fois l'adresse physique et Internet de la machine de destination.

II.2.2.3. La résolution par liaison dynamique

Ethernet a des adresses physiques de 48 bits affectés par les vendeurs pendant la fabrication de l'interface. En remplaçant une interface, l'adresse physique de la machine change. En plus, comme l'adresse Ethernet est de 48 bits, on ne peut pas l'encoder en une adresse Internet de 32 bits.

Les concepteurs d'Internet ont trouvé une solution créative à ce problème de résolution d'adresse pour des réseaux de type Ethernet. La solution permet d'ajouter des nouvelles machines au réseau sans code de recompilation, mais ne nécessite pas la maintenance d'une base de données centralisée. Pour éviter la maintenance d'une table de correspondance, ils ont choisi d'utiliser un protocole de bas niveau pour relier les adresses dynamiquement. L'idée est simple: quand un hôte A veut résoudre l'adresse Internet I_B , il diffuse un paquet spécial qui demande au hôte ayant cette adresse Internet de répondre avec son adresse physique, P_B . Tous les hôtes, y compris B, reçoivent la requête, mais seulement le hôte B reconnaît son adresse Internet et envoie une réponse qui contient son adresse physique. Quand A reçoit la réponse, elle retient l'adresse physique de B et l'utilise pour envoyer le paquet Internet directement à B.

Ainsi, le protocole de résolution d'adresses (ARP) permet à un hôte de trouver l'adresse physique d'un autre hôte sur le même réseau physique en donnant seulement l'adresse Internet de ce dernier.

II.2.2.4. La sauvegarde des correspondances des adresses

Il peut paraître anormal que A diffuse un message pour dire à B "comment je peux vous atteindre" au lieu de diffuser directement le paquet qu'elle veut transmettre. Mais il y a une raison importante pour cet échange. La diffusion coûte cher si elle est utilisé chaque fois qu'une machine doit transmettre un paquet puisqu'elle nécessite le traitement d'un paquet de diffusion. Pour réduire les coûts de communication, les hôtes qui utilisent les requêtes ARP gardent les dernières correspondances d'adresses (Internet-physique) dans une cache, de façon à ce qu'ils n'utilisent pas ARP de façon répétitive. Quand le hôte reçoit une réponse ARP, il enregistre le résultat dans cet endroit pour une utilisation ultérieure. Quand il veut envoyer un paquet à une destination quelconque, le hôte va d'abord voir dans cet endroit si la correspondance existe avant d'envoyer une requête ARP. S'il trouve la correspondance voulue, il n'envoie pas le message de requête sur le réseau.

On peut effectuer plusieurs raffinement à ce procédé. Premièrement, si un hôte A doit utiliser un ARP pour envoyer un message au hôte B, il y a une grande probabilité que le hôte B va répondre à A dans un futur proche. Si on anticipe les besoins de B, on peut éviter du trafic

supplémentaire en incluant une correspondance d'adresses de A quand celle-ci envoie une requête à B. Deuxièmement, comme A diffuse sa requête initiale, toutes les machines sur le réseau la reçoivent et peuvent l'extraire et l'enregistrer chez eux. Troisièmement, quand une nouvelle machine vient s'ajouter au réseau, on peut éviter que tous les autres machines exécutent un ARP en diffusant la nouvelle paire (correspondance des adresses).

Ainsi, dans chaque message ARP diffusé on trouve l'adresse Internet de l'émetteur et son adresse physique correspondante; le récepteur met à jour les informations de correspondances d'adresses dont il dispose avant de traiter un paquet ARP.

II.2.2.5. La relation entre ARP et les autres protocoles

ARP fournit un mécanisme de transformation des adresses Internet en adresses physiques; ce mécanisme n'est dipsonible que pour certains réseaux. Il serait inutile si tous les interfaces de réseaux comprenaient les adresses Internet. ARP impose un nouveau système d'adressage dépassant tout ceux utilisés par les machines.

Ainsi, ARP est un protocole de bas niveau qui cache l'adressage physique du réseau, pour nous permettre d'affecter des adresses Internet de notre choix à chaque machine. Il est considéré comme une partie du système physique du réseau, et non comme une partie des protocoles Internet.

II.2.2.6. L'implémentation de ARP

ARP est divisé en deux parties. Une partie qui détermine les adresses physiques quand on envoie des paquets, et l'autre répond aux requêtes des autres machines. La résolution des adresses des paquets transmis semble assez simple, mais les petits détails compliquent l'implémentation.

Le hôte consulte la table de correspondance des adresses. S'il trouve l'adresse physique correspondante à l'adresse Internet en question, il l'extrait, place les données dans la trame en utilisant cette adresse et envoie la trame. S'il ne trouve pas la correspondance, il doit diffuser une requête ARP et doit attendre une réponse.

La diffusion d'une requête ARP pour trouver une correspondance d'adresses Internet peut devenir complexe. La machine de destination peut être hors d'usage ou encombrée. Dans ce cas, elle ne peut pas traiter la requête et donc l'émetteur ne reçoit pas une réponse ou en reçoit une mais en retard. La requête ARP diffusée peut être également perdue.

Le hôte doit ranger, entre temps, les paquets à émettre pour qu'ils soient transmis une fois l'adresse a été résolue. En fait, le hôte doit décider s'il doit établir de requête ARP multiple ou pas, il doit savoir également s'il est permis d'envoyer plusieurs requêtes. Le hôte ne doit pas diffuser plusieurs requêtes ARP vers une adresse Internet de destination.

La valeur cachée peut être dépassée rendant ainsi le succès de la transmission impossible.

La deuxième partie du code ARP s'occupe des paquets ARP qui arrivent du réseau. Sa cache est examinée pour voir si elle a déjà une entrée pour l'émetteur et si c'est le cas, cette entrée

sera mise à jour en copiant l'adresse physique de l'émetteur du paquet. Le paquet sera traité par la suite.

Le hôte s'occupe de deux types de paquets ARP reçus. Il doit examiner les paquets reçus des requêtes ARP. Ces paquets doivent contenir la destination d'une requête d'une autre machine. Dans ce cas, une réponse est préparée et envoyée par le mécanisme ARP. Cette réponse contient son adresse physique. Sinon, le paquet doit contenir une demande de correspondance avec une autre machine et peut donc être ignoré. Dans ce deuxième cas, quand une réponse ARP arrive et selon l'implémentation, une entrée de cache est créée puis remplie par la correspondance Internet-physique de l'émetteur avant le traitement. Si les réponses envoyées concernent une requête, les paquets en attente seront transmis. Si le hôte ne s'intéresse pas à la réponse il arrête simplement le traitement du paquet.

II.2.3. La détermination de l'adresse Internet au démarrage (RARP)

II.2.3.1. Introduction

Habituellement l'adresse Internet d'une machine se trouve rangée sur le disque. Le système d'exploitation, au démarrage, va charger ces informations qui se trouve sur le support auxiliaire. Le problème qui se pose est de savoir comment une machine sans disque peut déterminer sa propre adresse Internet ? Le problème est critique pour les stations de travail sans disque qui utilisent des adresses Internet pour communiquer avec le serveur de fichiers. Ces machines utilisent le protocole RARP (pour *Reverse Address Resolution Protocol*) pour résoudre ce problème et ainsi obtenir une adresse.

L'idée de trouver une adresse Internet est assez simple: la machine sans disque envoie une demande à un serveur et attends une réponse de ce dernier. On suppose que le serveur a un disque sur lequel il dispose d'une base de données des adresses Internet. Dans la demande, la machine qui veut connaître son adresse Internet doit s'identifier d'une manière unique de façon que le serveur puisse chercher l'adresse Internet correcte et envoyer une réponse. Les deux machines utilisent les adresses physiques du réseau durant leur bref communication. La machine sans disque ne connaît pas l'adresse physique du serveur, elle diffuse simplement une requête à toutes les machines sur le réseau local. Un ou plusieurs serveurs répondent à cette requête.

Quand une machine sans disque diffuse sa requête, elle doit s'identifier d'une façon unique. Une information que le système d'exploitation peut diffuser et qui peut identifier d'une manière unique la machine sur laquelle il s'exécute peut être n'importe quel information sur l'identification du matériel. Le numéro de série du CPU, par exemple, suffit. Mais une telle information peut être difficile à obtenir, et la longueur ou le format peut varier d'un CPU à un autre.

II.2.3.2. Reverse Address Resolution Protocol (RARP)

Une autre information qui peut identifier d'une manière unique une machine et qui est toujours disponible est son adresse physique de réseau. L'utilisation de cette information a deux avantages:

1. cette adresse est toujours disponible puisqu'on peut l'obtenir de l'interface réseau du matériel sans être lié au système d'exploitation;
2. comme l'information d'identification dépend du réseau et non pas du modèle ou du vendeur du CPU, toutes les machines sur un réseau donné vont fournir des identifiants uniques et uniformes.

Ainsi, le problème devient une résolution d'adresses à l'envers: étant donné une adresse physique du réseau, un serveur la transforme en une adresse Internet.

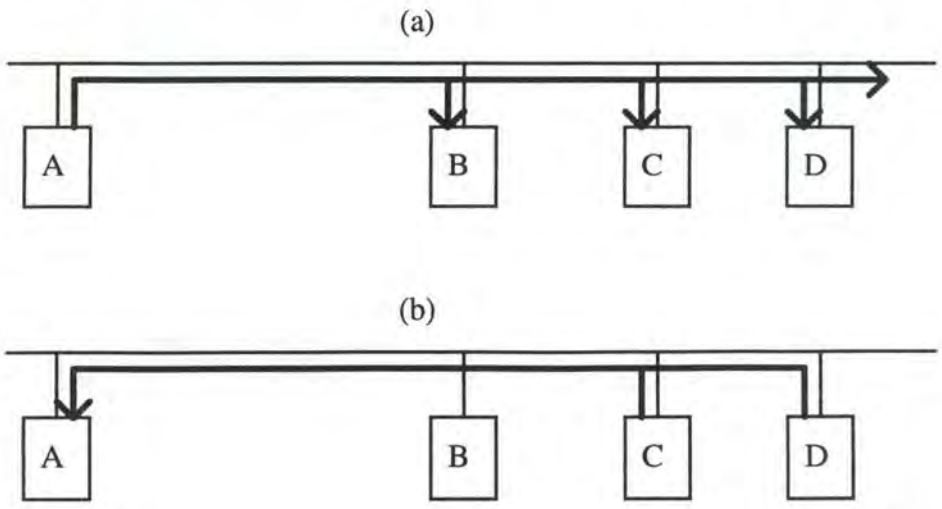
Le protocole que les machines sans disque utilise pour communiquer avec un serveur, qui peut leur fournir les adresses Internet, est appelé le *Reverse Address Resolution Protocol* (RARP). C'est une adaptation du protocole ARP, qu'on a développé au début du chapitre, il utilise le même format de message. Le message RARP envoyé demande une adresse Internet est plus général que ce qu'on a décrit: il permet a une machine de demander assez facilement n'importe quelle autre adresse Internet et plusieurs types de réseaux physiques.

Comme ARP, le message RARP est envoyé d'une machine à une autre encapsulé dans la partie de données d'une trame Ethernet. Une trame Ethernet qui transporte la requête RARP a le préambule habituel, les adresses Ethernet source et destination, et le champ type du paquet au début de la trame. Le type de la trame contient une valeur qui identifie le contenu de la trame comme étant une requête RARP. La partie de donnée de la trame contient le message RARP de 28 octets.

On peut illustrer la façon dont un hôte utilise RARP: l'émetteur diffuse une requête RARP qui se spécifie comme étant la machine cible et fournit son adresse physique de réseau dans le champ matériel cible (*target hardware*). Toutes les machines sur le réseau reçoivent la requête, mais seulement ceux qui sont autorisées de fournir le service RARP traitent la requête et envoient une réponse; de telles machines sont des serveurs RARP. Pour que le mécanisme RARP réussisse, le réseau devrait contenir au moins un serveur RARP.

La figure II.2.3.2. ci-dessous illustre l'utilisation de RARP:

Figure II.2.3.2. Exemple d'échange utilisant le protocole RARP:



En (a) le hôte A diffuse une requête RARP en se spécifiant comme étant cible, et en (b) ces machines autorisées à fournir le service RARP répondent directement à A.

Les serveurs répondent aux requêtes en remplissant le champ adresse cible du protocole, en changeant le type de message de requête à réponse et en envoyant la réponse directement à la machine qui a fait la requête. Cette machine reçoit des réponses de tous les serveurs RARP, même si la première suffit.

Toutes les communications entre le hôte qui cherche son adresse Internet et le serveur qui le fournit doivent être effectuées seulement sur le réseau physique. En plus, le protocole permet à un hôte de demander une cible arbitraire. Ainsi, l'émetteur fournit son adresse du matériel séparé de l'adresse cible du matériel, et le serveur est prudent quand il envoie la réponse en utilisant l'adresse du matériel de l'émetteur.

Avec Ethernet, le champ pour l'adresse du matériel de l'émetteur peut paraître redondant car l'information est aussi contenue dans l'en-tête de la trame Ethernet. Cependant, le technologie Ethernet ne donne pas, généralement, un accès au système d'exploitation à l'en-tête physique de la trame.

II.2.3.3. Les problèmes de transactions avec RARP

Comme toutes les autres communications réseau, les requêtes RARP peuvent être perdues ou corrompues. Le protocole RARP utilise seulement le réseau physique, son logiciel doit, donc, gérer les délais d'attente et les retransmissions. Généralement, RARP est seulement utilisé sur des réseaux locaux tel que Ethernet, où la probabilité d'un échec est faible. Cependant, si le réseau n'a qu'un seul serveur RARP, il n'est pas capable de manipuler le chargement, ainsi les paquets pourraient être perdus.

Plusieurs machines sans disque comptent sur RARP pour la mise en route et peuvent choisir de réessayer indéfiniment jusqu'à ce qu'ils reçoivent une réponse. D'autres implémentations annoncent un échec après seulement quelques essais pour éviter ainsi d'inonder le réseau avec un trafic de diffusion inutile. Sur Ethernet, la défaillance du réseau est moins probable qu'une surcharge du serveur. Si la retransmission RARP était plus rapide, on pourrait avoir des effet indésirable comme celui de l'inondation de serveurs, déjà encombrés, avec plus de trafic. Ainsi, un grand retard assure que les serveurs ont suffisamment de temps pour satisfaire la requête et renvoie une réponse.

II.2.3.4. Les serveurs RARP

Le principal avantage d'avoir plusieurs machines fonctionnant comme des serveurs RARP est de rendre le système plus fiable. Si des serveurs sont en panne ou sont lourdement chargés, d'autres peuvent répondre à la requête. Ainsi, il est très probable que le service sera disponible. Cependant, l'inconvénient d'utiliser plusieurs serveurs est que quand une machine diffuse une requête RARP, le réseau devient surchargé puisqu'ils vont tous tenter de répondre. Sur un Ethernet, par exemple, en utilisant plusieurs serveurs RARP, la probabilité d'une collision augmente.

Il y a deux possibilités pour arranger le service RARP de façon à le rendre disponible et fiable sans contracter des coûts de plusieurs réponses simultanées. Ces deux possibilités ont pour but

de retarder les réponses. Dans la première solution, un serveur principale est affecté aux machines qui font des requêtes RARP. Dans le cas normal, seul le serveur principal de la machine qui répond à sa requête RARP. Tous les autres serveurs reçoivent la requête et enregistrent seulement son heure d'arrivée. Si le serveur principal est non disponible, la machine émettrice va dépasser son délai d'attente de la réponse et va ensuite rediffuser la requête. Lorsque un autre serveur (non principal) reçoit une copie de la requête RARP dans le premier intervalle de temps, il répond.

La deuxième solution utilise un plan similaire mais tente d'éviter la transmission simultanée de la réponse des autres serveurs. Chaque machine, non principale, qui reçoit une requête calcule un délai aléatoire et ensuite envoie une réponse. Normalement, le serveur principal répond immédiatement, et des réponses successives sont retardées de façon à ce que la probabilité qu'ils arrivent en même temps est faible. Quand le serveur principal est non disponible, la machine émettrice subit des petits retards avant de recevoir une réponse.

Chapitre II.3.

DESCRIPTION ET FONCTIONNEMENT DE LA COUCHE IP

II.3.1. Objectifs et problèmes

Comme nous l'avons vu, l'Internet TCP/IP est constitué de trois ensembles de services superposés sous forme de couches dépendantes les unes des autres. En bas de la structure, on trouve la couche IP. Celle-ci fournit un service de livraison non fiable et sans connexion (*connectionless*). Dans la couche de dessus, on trouve le service TCP qui est fiable et qui fournit une plate forme de haut niveau de laquelle dépendent les programmes d'applications. Ceux-ci constituent la couche la plus élevé.

Le logiciel d'Internet est conçu autour de ces trois couches conceptuelles, qui sont arrangées hiérarchiquement, son succès résulte de cette architecture robuste et adaptable. Cette architecture présente des possibilités de remplacement de certains services sans toucher au reste.

Le protocole qui définit la non fiabilité et le mécanisme de livraison sans ouverture de connexion est appelé "*Internet protocole*", habituellement référencé par ses initiaux IP. Le protocole définit l'unité de base de transfert de données à travers le réseau TCP/IP, il spécifie dès lors le format exact des données qui vont être transmis. Le logiciel du protocole IP exécute une fonction de routage pour choisir un chemin par lequel les données doivent être envoyées.

En plus des spécifications des formats des données et du routage, il dispose d'un ensemble de règles qui expliquent la non fiabilité de la livraison des paquets. Ces règles définissent, la manière avec laquelle les hôtes et les passerelles devraient traiter les paquets, comment et quand les messages d'erreurs devraient être générés, et quelles sont les conditions dans lesquelles les paquets devraient être écartés.

Ce service est appelé "*connectionless*" car chaque paquet est traité indépendamment de tous les autres, il est donc envoyé sans ouverture de connexion. Les paquets envoyés d'une machine à une autre peuvent passer par différents chemins sur lesquelles quelques uns d'entre eux peuvent se perdre. Le logiciel d'Internet tente de livrer les paquets en faisant de son mieux (*best-effort delivery)* et donc il n'écarte jamais un paquet délibérément; le problème de non

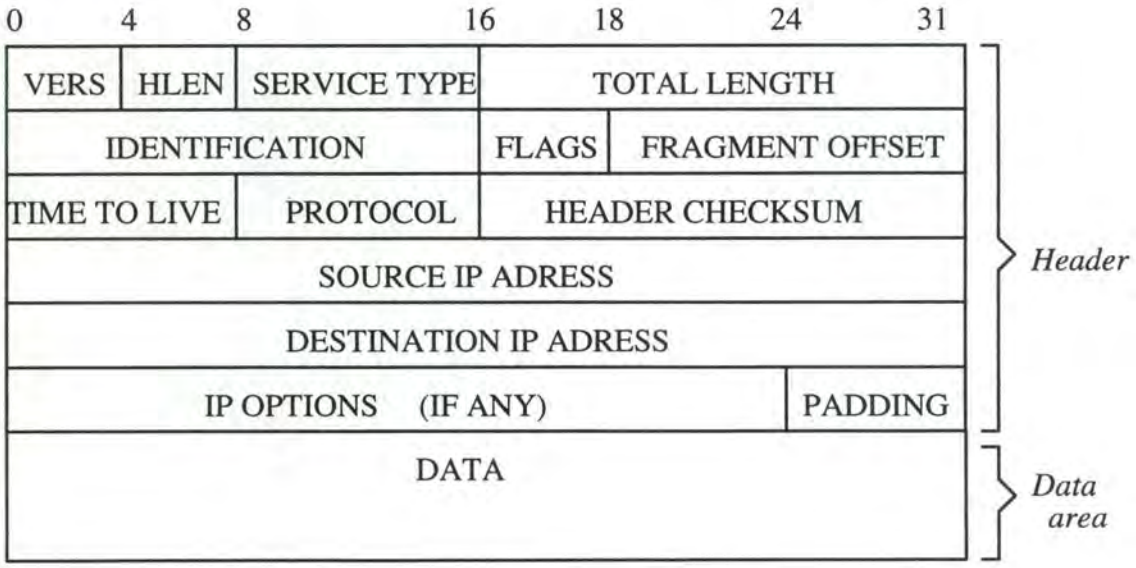
fiabilité est causé généralement par l'épuisement des ressources ou par une panne dans le réseau.

II.3.2. La structure statique : l'unité de transfert

L'unité de transfert de données de base est le datagramme d'Internet ou tout simplement datagramme. Comme pour les trames du réseau physique, un datagramme est divisé en deux parties: l'en-tête (*header*) et la zone de données (*data area*). L'en-tête, comme pour la trame, contient les adresses de la machine source et de destination et un champ pour spécifier le type de l'information du datagramme. Mais la différence avec une trame est que l'en-tête d'un datagramme contient les adresses IP cependant l'en-tête de la trame contient l'adresse physique.

II.3.2.1. Le datagramme

Comme le traitement des datagrammes se fait par le logiciel d'Internet correspondant à la couche IP, le contenu et le format du datagramme ne dépendent plus des machines. Ce format est le suivant:



II.3.2.2. Formats et contenus des champs

a/ <u>VERS:</u>

Ce champ de 4 bits contient la version du protocole IP utilisé pendant la création du datagramme. On utilise ce champ pour que les machines émettrices et réceptrices et les passerelles parcourues se mettent d'accord sur le format du datagramme. Tous les logiciels IP doivent vérifier le champ VERS avant de traiter un datagramme.

Si les standards changent, les machines rejetteront les datagrammes dont la version du protocole diffère de la leur.

b/ HLEN:

Ce champ est également d'une longueur de 4 bits, il donne la longueur de l'en-tête du datagramme en mots de 32 bits. Tous les champs de l'en-tête ont la même longueur excepté le champ "IP OPTION" et le champ correspondant "PADDING".

Un en-tête qui mesure 20 octets n'a pas les champs "IP OPTION" et "PADDING", son champ HLEN est donc égale à 5.

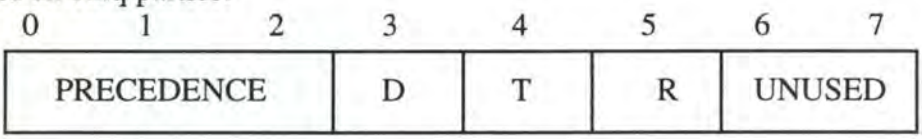
c/ TOTAL LENGTH:

Ce champ donne la longueur en octets de tout le datagramme IP (*header* + *data*). La taille de la zone de données peut donc être calculée en retirant HLEN du TOTAL LENGTH. Comme le champ TOTAL LENGTH a une longueur de 16 bits, la taille maximale possible d'un datagramme IP est de 65535 octets. Dans la plupart des applications cette limitation est acceptable. Cette limite pourrait devenir importante si dans l'avenir il y auront des réseaux à grande vitesse dont les paquets de transmission de données dépassent 65535 octets.

d/ SERVICE TYPE:

Ce champ a une longueur de 8 bits, il spécifie la façon dont le datagramme doit être manipulé.

Il est divisé en cinq parties:



d-i / PRECEDENCE:

Ce sous-champ a une longueur de 3 bits, il spécifie la priorité du datagramme, sa valeur varie de 0 (priorité normale) à 7 (commandes réseau), permettant ainsi à l'émetteur d'indiquer l'importance de chaque datagramme.

Malgré que la majorité des hôtes et des passerelles ignorent ce champ, ce concept peu être utilisé puisqu'il offre un mécanisme de contrôle de l'information en mettant des priorités sur les données. Par exemple, si tous les hôtes et passerelles utilisent ce champ de priorité, il serait possible d'implémenter des algorithmes capables de contrôler l'encombrement des réseaux. Cet encombrement d'affectera pas ces algorithmes.

d-ii/ D, T et R:

Chacun de ces sous champ a une longueur d'un bit. Ils spécifient le type de transport désiré par le datagramme. Les spécifications du type de transport n'est donc qu'une suggestion ou une indication à l'algorithme de routage, qui va l'aider à choisir le chemin optimal. Ces algorithmes se basent sur leurs connaissances des technologies du matériel disponibles sur ces chemins.

Un réseau Internet ne garantit donc pas le type de transport demandé car il pourrait ne pas y avoir un chemin vers la destination, qui possède ces propriétés. Quand les trois bits sont mis à

1; pour le champ D, la durée de transmission du datagramme doit être brève (*low delay)*; pour T, le débit de transmission doit être très élevé; et enfin pour R, une grande fiabilité du support est exigée. Si une passerelle connaît plus d'un chemin pour atteindre une destination, elle pourrait utiliser le champ type de transport pour sélectionner une route dont les caractéristiques sont les plus proches de celles demandées.

Le contrôle de fragmentation:

Trois champs de l'en-tête du datagramme s'occupent du contrôle, de la fragmentation et de l'assemblage, l'"IDENTIFICATION", le "FLAG", et le "FRAGMENT OFFSET":

e/ IDENTIFICATION:

Ce champ contient un entier unique qui identifie le datagramme et qui sert, par exemple, en cas de fragmentation. Comme les passerelles, par exemple, peuvent fragmenter un datagramme, ils copient la majorité des champs de l'en-tête du datagramme dans chaque fragment. Le champ identification doit donc être copié sur chaque fragment, pour permettre à la destination de savoir si le fragment appartient ou non à un datagramme. Quand le fragment arrive, la destination utilise le champ identification avec l'adresse source du datagramme pour identifier le datagramme.

Les ordinateurs qui envoient des datagrammes doivent gérer une valeur unique du champ identification pour chaque datagramme. Une technique utilisée par le logiciel IP est de tenir un compteur en mémoire, qui est incrémenté chaque fois qu'un nouveau datagramme est créé, et assigne le résultat au champ identification.

f/ FRAGMENT OFFSET:

Les grands paquets sont généralement fragmentés à l'émmission et rassemblés à l'arrivée. Les fragments obtenus ont exactement le même format qu'un datagramme complet. Pour un fragment, le champ FRAGMENT OFFSET indique l'emplacement de ce fragment dans le datagramme initial, mesuré en unités de 8 octets, et commençant par l'emplacement numéro zéro. Pour rassembler le datagramme, la destination doit recevoir tous les fragments commençant par le fragment qui possède le numéro d'emplacement zéro jusqu'au fragment ayant le numéro le plus élevé. Les fragments n'arrivent pas nécessairement dans l'ordre, et il n'y a aucune communication entre la passerelle qui fragmente le datagramme et la destination qui essaye de les rassembler. Si un des fragments se perd, le datagramme entier doit être écarté.

g/ FLAGS:

Ce champ est composé de 3 bits, il s'occupe du contrôle de la fragmentation. Son premier bit indique si le datagramme peut être fragmenté ou pas, il est appelé "le bit de non fragmentation", s'il est mis à 1, le datagramme ne doit pas être fragmenté. Les premiers bits de droite (d'ordre faible) du FLAG indiquent le dernier fragment (le fragment qui a le plus grand OFFSET). Comme le champ LENGTH du l'en-tête du fragment s'applique à la taille du fragment et non à celle du datagramme initial, la destination ne peut pas utiliser cette longueur pour vérifier si elle a collecté tous les fragments. Ainsi elle a besoin du bit "encore des fragments" qui marque la fin du datagramme initial.

h/ TIME TO LIVE:

Ce champ indique l'intervalle de temps maximal, en secondes, autorisé au datagramme pour rester dans le réseau Internet. En principe, l'idée est à la fois simple et importante: chaque fois qu'un hôte transmet un datagramme, il fixe un temps maximal durant lequel un datagramme devrait survivre. Chaque passerelle, tout le long du chemin entre la source et la destination, traite le datagramme. Elle vérifie le temps restant et décrémente ce champ de 1, si le "*time to live*" atteint le zéro, le datagramme serait écarté.

Ainsi, les datagrammes ne pourront plus circuler indéfiniment dans le réseau même si la table de routage est corrompue.

Il est difficile d'estimer le temps exact car les passerelles ne connaissent pas habituellement le temps de transit dans le réseau. Pour simplifier le traitement, les hôtes et les passerelles supposent que chaque transfert de réseau dure une unité de temps. Ainsi, ils doivent décrementer le champ "*time to live*" d'une unité chaque fois qu'ils traitent un en-tête.

i/ PROTOCOL:

Ce champ est analogue à celui du type de la trame d'Ethernet. Les protocoles de haut niveau utilisent "PROTO" pour indiquer le format et le contenu de la donnée en identifiant le type de protocole. L'organisation entre le protocole de haut niveau et la valeur entière utilisée dans ce champ doit être gérée par une autorité centrale pour garantir la standardisation à travers tout le réseau Internet.

j/ HEADER CHECKSUM:

Ce champ assure l'intégrité de la valeur de l'en-tête. La somme de contrôle IP ou "IP *checksum*" est formé en traitant l'en-tête comme une séquence d'entiers de 16 bits. Ces entiers sont additionnés en utilisant le complément arithmétique, et ensuite on prend le complément du résultat. Pour des besoins de calcul de la somme de contrôle, ce champ est supposé contenir zéro.

La somme de contrôle se trouve dans l'en-tête séparée des données. L'en-tête occupe d'habitude moins d'octets que les données, la séparation des sommes de contrôle réduit le temps de traitement dans les passerelles qui ont seulement besoin de calculer les sommes de contrôle des en-têtes. La séparation permet également au protocoles de haut niveau de choisir leur propre formules pour la somme de contrôle. En plus, les protocoles de haut niveau sont forcés d'ajouter leur propre somme de contrôle ou risquent d'avoir des données corrompues passées sans être détectées.

k/ SOURCE IP ADDRESS et DESTINATION IP ADDRESS:

Ces champs contiennent des adresses Internet de 32 bits, respectivement de l'émetteur du datagramme et du récepteur .

l/ DATA:

Ce champ contient les données du datagramme. La longueur de ce champ dépend de ce qui va être transmis dans le datagramme.

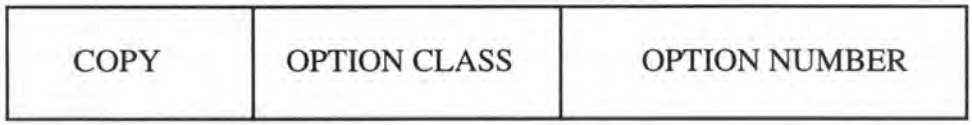
m/ IP OPTION / PADDING:

Le champ "IP OPTION", discuté ci-dessus, a une longueur variable. Le champ "PADDING" dépend de l'option sélectionnée, il représente les bits contenant zéro dont on peut avoir besoin pour assurer que l'en-tête Internet s'étend à un multiple exact de 32 bits. La longueur du champ de l'en-tête étant spécifiée en unités de mots de 32 bits.

m-i / Les options du datagramme d'Internet:

Le champ "IP OPTION" qui suit le champ adresse de destination n'est pas exigé dans chaque datagramme; les options sont inclus généralement pour les tests ou la mise au point des réseaux. La longueur de ce champ varie suivant les options choisies.

Quelques options ont une longueur d'un octet, d'autres ont des longueurs variables. Quand il y a des options présentes dans un datagramme, elles apparaissent comme étant contiguës, elles ne sont pas séparées par des séparateurs spéciaux. Chaque option consiste en un code option d'un octet, une longueur d'un octet, et un ensemble d'octets de données pour l'option. L'octet du code option est divisé en trois champs comme la figure ci-dessous le montre:



Les champs consistent en un drapeau d'une longueur d'un bit appelé COPY qui contrôle le traitement des options par les passerelles durant la fragmentation, quand ce bit est à 1, il indique que l'option devrait être copiée dans tous les fragments, s'il est à 0, l'option est seulement copiée dans le premier fragment; d'un deuxième champ de 2 bits appelé OPTION CLASS, et d'un troisième champ de 5 bits appelé OPTION NUMBER, ces deux champs indiquent la classe générale de l'option et donnent une option spécifique dans cette classe.

La figure ci-dessous montre comment ces classes sont assignées:

Option Class	Explications
0	contrôle de datagramme ou de réseau
1	réservé pour des utilisations futures
2	mesures et mises au point
3	réservé pour des utilisations futures

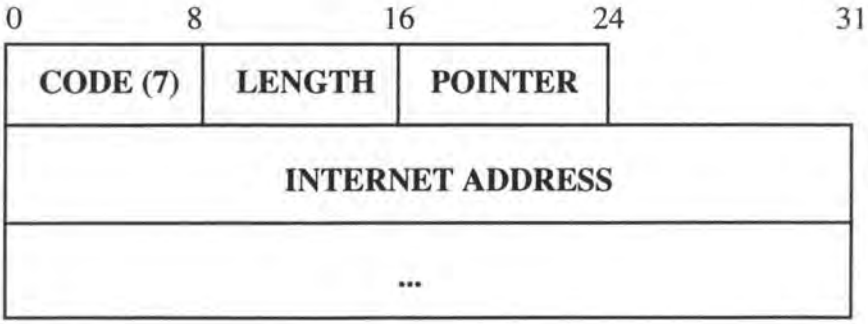
Les options possibles qui peuvent accompagner un datagramme IP sont données dans la figure ci-dessous, pour chaque OPTION CLASS on donne sa valeur OPTION NUMBER. On remarque que la majorité des options sont utilisées pour des buts de contrôle:

Option Class	Option Number	Length	Description
0	0	1	Fin de la liste des options. Utilisé si les options ne se terminent pas à la fin du datagramme.
0	1	1	Pas d'opérations.
0	2	11	Des restrictions de manipulations et de sécurité.
0	3	var	Perte du routage source. Utilisée pour router le datagramme à travers un chemin spécifié.
0	7	var	Enregistrement des routes. Utilisée pour tracer une route.
0	8	4	Identificateur de flux. Utilisée pour transporter un identificateur d'un flux SATNET.
0	9	var	Routage stricte de la source. Utilisée pour router les datagrammes à travers un chemin spécifique.
2	4	var	Marqueur de temps d'Internet. Utilisée pour enregistrer les marques de temps le long du chemin.

m-ii/ Les options d'enregistrement des routes:

Les options de routage et de marque de temps sont les plus importants car ils offrent un moyen de contrôler comment les passerelles routent les datagrammes. Cette option permet à la source de créer une liste vide d'adresses IP et s'arrange pour que chaque passerelle qui manipule le datagramme ajoute son nom à la liste.

Le format de l'option enregistrement de la route est présenté ci-dessous:



Comme décrit ci-dessus, le champ CODE contient le numéro et classe d'option (7 pour l'enregistrement de la route). Le champ LENGTH indique la longueur totale de l'option comme elle apparaît dans le datagramme IP, les trois premiers octets compris.

Le champ INTERNET ADDRESS dénote la zone réservé pour les adresses Internet, et un champ POINTER qui donne le complément dans l'option du prochain emplacement disponible.

Chaque machine qui manipule le datagramme ajoute son adresse à la liste d'enregistrement des routes. Pour cela, la machine compare d'abord le pointeur et le champ longueur. Si le

pointeur est plus grand que la longueur, la liste est alors pleine, et la machine expédie le datagramme sans rien insérer dans la liste. Si la liste n'est pas pleine, la machine insère son adresse de 4 octets et incrémente le pointeur de 4 unités.

m-iii/ Les options de routage de la source:

Une autre option que les constructeurs de réseaux trouvent intéressante est l'option routage de la source. L'idée est de fournir un moyen pour l'émetteur de dicter un chemin à travers Internet.

Par exemple, pour tester le débit d'un réseau donné, les administrateurs du système peuvent s'arranger pour forcer les datagrammes IP sur un de ces réseaux en utilisant le routage de la source.

En pratique, IP supporte deux formes de routage de la source. La première forme appelé le routage stricte de la source, qui comprend une séquence d'adresses Internet. C'est à dire que les adresses spécifient le chemin que le datagramme doit suivre pour atteindre sa destination. Le chemin entre deux adresses successives dans la liste doit consister en un seul réseau physique; une erreur résulte si une passerelle ne peut pas suivre le routage stricte de la source. L'autre forme, appelé la perte du routage de la source, comprend également une séquence d'adresses Internet. Elle indique au datagramme de suivre la séquence des adresses Internet, mais permet le saut de plusieurs réseaux entre les adresses de la liste.

Tout au long du chemin, les passerelles doivent mettre leurs adresses du réseau local dans la liste. Ceci est exigé par les deux options de routage. Ainsi, quand le datagramme arrive à sa destination, il contient une liste de tous les adresses visitées, exactement comme la liste produite par l'option enregistrement des routes.

Le format de l'option routage de la source est comme celle de l'enregistrement des routes. Chaque passerelle examine les champs du pointeur et de la longueur pour voir si la liste a été épuisée. Si oui, le pointeur sera plus grand que la longueur; la passerelle route le datagramme vers sa destination comme d'habitude. Si la liste n'est pas épuisé, la passerelle suit le pointeur, prend l'adresse Internet, la remplace avec l'adresse de la passerelle, et route le datagramme en utilisant l'adresse obtenue de la liste.

m-iiii/ L'option "Timestamp":

Cette option fonctionne comme celle de l'enregistrement des routes, cependant, elle contient une liste initialement vide et chaque passerelle visité tout le long du chemin entre la source et la destination va remplir un article de cette liste. Les entrées sont cependant constituées par l'heure et la date à laquelle la passerelle a manipulé le datagramme, exprimés en millisecondes à partir de minuit.

Ce "timestamp" a également une option qui permet à la source d'exiger de chaque passerelle d'insérer à la fois un "timestamp" et son adresse Internet dans la liste. D'habitude, cette option est utilisée pour contrôler la performance d'un réseau. Avec l'enregistrement des routes et les "timestamps", le récepteur peut savoir exactement quels sont les chemins que le datagramme a emprunté.

II.3.3. Fonctionnement dynamique

II.3.3.1. La fragmentation

Le réseau transporte les datagrammes d'une machine à l'autre. Pour que ce transport soit efficace, chaque datagramme doit être transmis dans une trame physique distincte, ainsi l'intégrité du datagramme est garantie. L'idée de transporter un datagramme dans une trame est appelé "encapsulation".

Le réseau ne reconnaît pas le format du datagramme et ne comprend pas l'adresse IP de destination; alors quand une machine envoie un datagramme IP à une autre, tout le datagramme sera mis dans la partie donnée de la trame. Dans le meilleur des cas, le datagramme IP s'adapte bien à la taille de la trame, ce qui rend la transmission efficace. Pour cela, la sélection de la taille maximale d'un datagramme IP doit bien correspondre à celle de la trame.

La quantité maximale de données qui doit être transférée dans une trame, est fixée par la technologie de commutation utilisée; par exemple pour Ethernet, la limite est de 1500 octets de données, alors que pour d'autres supports comme proNET-10 par exemple, elle peut atteindre 2044 octets par trame (Comer 88).

Ces unités de données s'appellent des MTU ou unités maximales de transfert du réseau. Les MTU peuvent être très petites, de l'ordre de 128 octets ou moins pour certaines autres technologies. Si on adapte la taille des datagrammes aux plus petites MTU possibles, le transfert deviendrait inefficace, surtout si ces datagrammes empruntent des réseaux qui peuvent acheminer des trames beaucoup plus larges. Cependant, si on utilise des datagrammes plus larges que le minimum des MTU dans un réseau Internet, le datagramme ne s'adapterait pas toujours à une trame.

Le logiciel de TCP/IP a choisit une taille optimale du datagramme, il s'est également arrangé pour diviser les larges datagrammes en plus petites parties. Ces petites parties sont appelées fragments et le processus de cette division est appelé fragmentation.

Ces fragmentations se produisent habituellement dans les passerelles entre la source et la destination. La passerelle reçoit des paquets dans des MTU de grande taille et doit les router sur un autre réseau dans lequel les MTU sont de taille plus petites. Ces datagrammes sont fragmentés et la taille des fragments résultants doit être multiple de 8 octets. En choisissant le multiple de 8 octets le plus proche de la taille du MTU, la division du datagramme en fragments de même taille n'est plus garantie; le dernier fragment est souvent plus petit que les autres. Les fragments doivent être rassemblés pour reproduire une copie complète du datagramme originale avant d'être traité par la destination. La fragmentation et l'assemblage s'effectuent automatiquement sans que l'émetteur prenne des dispositions précises.

Un datagramme ayant, par exemple, 1400 octets de données, est divisé en trois fragments de 620 octets, chaque fragment contient un en-tête qui reprend la majorité des informations contenues dans l'en-tête du datagramme initial (excepté le bit du champ "FLAG" qui indique que c'est un fragment), suivi par autant de données que le fragment peut transporter; la longueur totale ne doit pas dépasser celle du MTU du réseau.

Dans le réseau TCP/IP, une fois que le datagramme a été fragmenté, les fragments se propagent comme des datagrammes séparés jusqu'à leur destination où ils doivent être assemblés.

La fragmentation peut générer des problèmes. D'abord les fragments peuvent traverser des réseaux différents qui peuvent avoir des MTU de taille très large ce qui diminue l'exploitation optimale des réseaux; en plus si un des fragments est perdu, le datagramme ne peut plus être assemblé. En fait, la machine réceptrice déclenche une horloge d'assemblage quand elle reçoit le premier fragment, si l'intervalle de temps critique est dépassé avant que tous les autres fragments arrivent, la machine réceptrice écarte les fragments reçus sans traiter le datagramme. Ainsi, la probabilité de perte de datagrammes augmente quand il y a une fragmentation car la perte d'un seul fragment implique la perte de tout le datagramme.

Malgré ces problèmes, la technique de l'assemblage à la destination fonctionne bien, elle permet ainsi aux fragments d'être routés indépendamment et donc ils n'ont plus besoin de passerelles intermédiaires pour rassembler les fragments.

II.3.3.2. Le routage des datagrammes

Les services Internet sont basés sur un système non fiable de livraison de paquets et sans ouverture de connexion . Ce système a comme unité de transfert de base le datagramme IP. Celui-ci est routé par des passerelles vers sa destination finale. L'algorithme de routage constitut l'aspect opérationnel de l'IP alors que le format du datagramme est l'aspect statique.

Le routage peut être difficile surtout avec des machines connectées par plusieurs réseaux. Le code de routage doit sélectionner un réseau, sur lequel le datagramme doit être envoyé.

L'idéal est que le logiciel de routage, pour sélectionner le meilleur chemin, examine la charge du réseau, la longueur du datagramme, ou le type de service spécifié dans l'en-tête de celui-ci. Mais malheureusement, la plupart de ces logiciels sont beaucoup moins sophistiqués, ils sélectionnent les routes en se basant sur des suppositions fixes des chemins les plus courts.

Nous allons converger sur le routage des protocoles Internet. Le routage IP est analogue à celui des réseaux physique, cependant il se fait à un niveau plus haut que ce dernier.

Pour bien comprendre le routage IP, revenons à l'architecture d'Internet. On sait qu'Internet est composé de plusieurs réseaux physiques interconnectés par des ordinateurs appelés des passerelles. Les hôtes quant à eux , sont connectés directement à un ou plusieurs réseaux physiques. Les hôtes et les passerelles participent dans le routage IP.

II.3.3.3. Les types de routage

Pour ce qui suit, on va distinguer les hôtes des passerelles, et on va supposer que les hôtes n'exècutent pas les fonctions des passerelles qui consistent à transférer les paquets d'un réseau à un autre.

On peut distinguer deux types de routage, le routage direct et indirect:

- le routage direct: la transmission d'un datagramme se fait directement d'une machine à une autre. Il n'y a pas de machines intermédiaires, car les deux machines se trouvent sur le même réseau. C'est la base sur laquelle fonctionnent toutes les communications Internet;

- le routage indirect: quand la destination n'est pas sur le même réseau, elle force l'émetteur de passer le datagramme à une passerelle pour la livraison.

II.3.3.3.1. Le routage direct

Une machine sur un réseau physique donné peut envoyer directement une trame à une autre machine sur le même réseau. Pour transférer un datagramme IP, l'émetteur met le datagramme dans une trame physique, transforme l'adresse IP en une adresse physique, et utilise le réseau pour la livraison. Ainsi, la transmission d'un datagramme IP entre deux machines sur un seul réseau physique n'implique pas de passerelles; la trame qui contient le datagramme est envoyée directement à la destination.

Pour voir si la destination se trouve sur un des réseaux connecté directement à la machine, l'émetteur extraît la partie concernant le réseau de l'adresse Internet de destination, et la compare à la partie réseau de sa propre adresse Internet. Si les deux parties coïncident, le datagramme peut être envoyé directement en utilisant l'adresse physique.

On peut ainsi voir les avantages de l'organisation des adresses Internet: les adresses de toutes les machines qui se trouvent sur le même réseau comprennent un identificateur de réseau commun, et comme l'extraction de cet identificateur peut être faite par quelques instructions, on peut tester si une machine peut être directement atteinte; ce qui est extrêmement efficace.

II.3.3.3.2. Le routage indirect

Le routage indirect est plus difficile que le routage direct car l'émetteur doit identifier une passerelle vers laquelle on peut envoyer le datagramme. La passerelle doit également retransmettre le datagramme sur le réseau de destination.

Supposons qu'on a un réseau Internet composé de plusieurs sous-réseaux interconnectés par des passerelles. Sil y'a seulement deux hôtes aux deux bouts, l'hôte émetteur va transmettre le datagramme vers la passerelle la plus proche. Cependant, comme tous ces sous-réseaux sont interconnectés, chacun d'entre eux doit avoir une passerelle qui le relie avec le reste des sous-réseaux.

Ainsi, l'hôte émetteur peut atteindre une passerelle en utilisant un seul réseau physique. Une fois que la trame atteint la passerelle, un programme va extraire le datagramme et une routine de routage va ensuite sélectionner la prochaine passerelle le long du chemin de la destination. Le datagramme est de nouveau placé dans une trame et envoyé au réseau suivant et donc vers une seconde passerelle, etc., jusqu'à ce qu'il peut être livré directement.

L'idée est donc que les passerelles forment une structure interconnectée de coopération, les datagrammes passent d'une passerelle à une autre jusqu'à ce qu'ils atteignent une passerelle qui peut les livrer directement à leur destination.

II.3.3.3.3. Les algorithmes de routage

a/ Le routage sur base de tables

L'algorithme de routage habituel emploie sur chaque machine une table de routage. Cette table contient des informations sur les destinations possibles. Si chaque table de routage contenait des informations sur chaque adresse de destination possible, il serait impossible de garder ces tables courantes. De plus, comme le nombre de destinations possibles est grand, les machines n'auraient pas suffisamment de place pour ranger toutes les informations.

Les passerelles routent les paquets sans connaître les détails concernant les hôtes et leurs environnements locaux. Heureusement, le plan de l'adresse IP rend un tel routage possible. Les passerelles n'ont pas besoin de connaître tous les détails tant qu'ils connaissent la structure d'interconnexion des réseaux et le réseau de destination auquel un datagramme doit être transmis.

Quand un datagramme arrive à une passerelle, le logiciel IP localise l'adresse de destination Internet et extrait la partie concernant le réseau. La passerelle utilise ensuite l'identificateur du réseau pour prendre les décisions de routage. L'utilisation des adresses du réseau de destination plutôt que des adresses du hôte de destination rend le routage efficace et garde une table de routage assez petite.

Typiquement, une table de routage contient des paires (N,G), où N est l'adresse du réseau Internet de destination, et G est l'adresse Internet d'une passerelle à laquelle les datagrammes doivent être envoyés, ils seront donc destinés aux réseaux N. Tous les passerelles listés dans la table de routage et utilisées par une machine M, doivent être reliées aux réseaux sur lesquels la machine M est directement connecté, rendant ainsi la liaison directe possible. Ainsi pour garder la taille de cette table minime, le logiciel de routage IP se base sur les adresses de réseaux de destination et non pas sur les adresses individuelles de hôtes.

Cependant, le fait de choisir des routes basés seulement sur des identificateurs de réseaux de destination peut avoir plusieurs conséquences.

Premièrement, dans la plupart des implémentations, ceci veut dire que tous les débuts de trafic pour un réseau donné prennent le même chemin. Comme résultat, même s'il existe plusieurs chemins, ils peuvent ne pas être utilisés simultanément. Aussi tous les types de trafic suivent le même chemin sans tenir compte des retards ou des débits des réseaux physiques.

Deuxièmement, la seule passerelle qui peut communiquer avec le hôte de destination est la dernière sur le chemin. Elle est la seule à savoir si le hôte existe et s'il est opérationnel. Ainsi, nous avons a besoin de construire un chemin vers cette passerelle pour envoyer à la machine source le rapport sur les problèmes rencontrés.

Troisièmement, car chaque passerelle va router le trafic de façon indépendante, le passage de ce trafic d'un hôte A vers un hôte B peut suivre des chemins totalement différents que celui du passage de B vers A. Pour cela, les passerelles doivent coopèrer pour garantir que cette communication à deux voies soit toujours possible.

On sait maintenant que IP route les datagrammes en se basant sur une table de routage, cette table de routage doit être initialisée ou mise à jour chaque fois que le réseau change. Pour cela il y a des protocoles qui permettent aux passerelles de garder les routes cohérentes. Il est donc important de savoir que IP se base sur les tables pour ses décisions de routage, et qu'en changeant ces tables on change les routes suivies par les datagrammes.

Ainsi, le routage IP se compose de décisions du type "vers où je vais envoyer le datagramme" en se basant sur son adresse de destination. La livraison directe implique l'encapsulation du datagramme dans une trame physique, la transformation de l'adresse Internet de destination en une adresse physique par l'intermédiaire du protocole ARP (*Address Resolution Protocol*), et l'envoi de la trame sur le réseau.

Pour l'algorithme de routage Internet orienté tables, les décisions de routage se basent sur les adresses de réseau de destination plutôt que sur les adresses des hôtes de destination, gardant ainsi la table de routage petite et efficace. Le routage par défaut sert à garder une petite table, surtout pour les hôtes qui peuvent accéder à une seule passerelle.

b/ Les routes par défaut

Une autre technique utilisée pour garder une table de routage minime est de consolider plusieurs entrées en un seul cas appelé "cas par défaut". L'idée est d'avoir un logiciel de routage IP qui cherche d'abord le réseau de destination dans la table de routage. Si aucune route n'apparaît dans la table, les routines de routages envoient le datagramme au cas par défaut.

Le routage par défaut est surtout utile pour les sites disposant d'un petit ensemble d'adresses locales et ayant une seule connexion avec l'Internet. Par exemple, les routes par défaut sont plus efficaces dans les machines hôtes qui sont reliés à un seul réseau physique et ne pouvant atteidre qu'une seule passerelle qui conduit au reste de l'Internet.

Le mécanisme de routage se fait en deux tests: un pour le réseau local et un par défaut qui pointe vers la seul passerelle possible. Même si le site contient peu de réseaux locaux, le routage est simple puisqu'il consiste en quelques tests pour les réseaux locaux et un par défaut pour toutes les autres destinations.

c/ Les routes spécifiques aux hôtes

Tous les routages sont basés sur les adresses de réseaux et pas sur des hôtes individuels. Malgré cela, la plupart des logiciels de routage IP fournissent des routes pour chaque hôtes. Ces hôtes doivent être spécifiées comme étant des cas spéciaux.

Ayant ces routes par hôtes, l'administrateur du réseau local aura plus de contrôle sur le réseau, ils peuvent également être utilisés pour le contrôle de l'accès. En mettant au point les

connections de réseaux ou les tables de routage, la capacité de spécifier une route spéciale pour une machine individuelle se révèle très utile.

d/ L'algorithme générale

En tenant compte de tout ce qu'on avait dit, l'algorithme de routage IP devient:

Algorithme

Routage_du_datagramme_IP (datagramme, table_de_routage)

```
Exraire l'adresse IP de destination, ID, du datagramme
Calculer l'adresse IP du réseau de destination, IN
Si IN correspond à une adresse réseau directement connectées
Alors Envoyer le datagramme à sa destination sur ce réseau;
        (Ceci implique que ID est transformé en une adresse physique,
        que le datagramme est encapsulé, et que la trame est transmise)
Sinon si ID apparait comme une route de hôte-spécifique
        Alors Router le datagramme comme spécifié dans la table;
        Sinon si IN apparait dans la table de routage
                Alors Router le datagramme comme spécifié dans la table;
                Sinon si une route par défaut a été spécifié
                        Alors Router le datagramme vers le "gateway" par défaut
                        Sinon déclarer une erreur de routage;
```

II.3.3.3.4. La manipulation des datagrammes à l'arrivée

Les paquets reçus sont également impliqués dans le routage. Quand un datagramme IP arrive à un hôte, il est livré au logiciel IP pour être traité. Deux cas se présentent: le datagramme a atteint sa destination finale, ou bien, il doit continuer son chemin. Dans le premier cas, l'IP doit passer le datagramme à l'application de destination pour le traitement. Dans le deuxième cas, l'IP retient le datagramme et le route en utilisant l'algorithme standard.

Une machine peut avoir plusieurs connexions physiques, chacune de ces connexions a sa propre adresse Internet. Quand un datagramme arrive, le hôte doit comparer l'adresse Internet de destination avec chacune de ses adresses de réseau. Si une d'entre elles corresponde à l'adresse de destination, il passe le datagramme à l'application de destination. S'il n'y a aucune correspondance, IP décremente le champ "*time-to-live*" de l'en-tête du datagramme, il l'écarte quand ce compte atteigne zéro sinon il le route.

Les passerelles doivent router les datagrammes reçus car c'est leur fonction principale. Plusieurs hôtes dotés d'une multitude de point de départs agissent comme des passerelles. Les hôtes qui ne sont pas désignés comme étant des passerelles ne doivent pas router les datagrammes qu'ils reçoivent; ils doivent les écarter. Il y a quatre raisons pour lesquelles ces hôte doivent s'abstenir d'exécuter les fonctions des passerelles:

1. Quand un tel hôte reçoit un datagramme destiné à d'autres machines, c'est qu'il y a eu un mauvais fonctionnement dans l'adressage d'Internet ou dans le routage. Le problème peut ne pas être révélé si le hôte prenait des actions correctives en reroutant le datagramme.

2. Le reroutage va causer un trafic inutile dans le réseau.

3. Les erreurs simples peuvent causer un chaos. Supposons que chaque hôte reroute le trafic, ce qui va se passer si une erreur déclenche la diffusion d'un datagramme qui est destiné à un hôte H, est que chaque hôte sur le réseau qui reçoit ce datagramme, le reroute vers H, celui-ci va être criblé par un grand nombre de copies.

4. Les passerelles utilisent un protocoles spéciale pour signaler les erreurs; les hôtes qui ne sont pas désignés pour servir comme passerelle ne signalent pas les erreurs.

Chapitre II.4.

LE PROTOCOLE ICMP

II.4.1. Objectifs et problèmes

II.4.1.1. Les objectifs

On a vu que le protocole Internet offre un service de livraison de paquets non fiable et sans connexion, et que les paquets traversent les passerelles pour arriver à leur hôte de destination. Si une passerelle ne peut pas router ou livrer un datagramme, ou si elle détecte des conditions inhabituelles, comme par exemple l'encombrement du réseau, elle informe les hôtes de ces conditions pour qu'ils puissent prendre des décisions pour résoudre le problème. Le mécanisme qu'utilisent les passerelles et les hôtes pour communiquer de telles informations de contrôle ou d'erreurs est spécifié dans le protocole des messages de contrôle Internet dit ICMP (pour *Internet Control Message Protocol*). Ce mécanisme est utilisé par les passerelles pour résoudre les problèmes de report de livraison, il est également utilisé par les hôtes pour tester si des destinations peuvent être atteintes.

ICMP fournit des messages pour réduire le taux de transmission ou flux à la source, des messages de redirection qui demandent à un hôte de changer ses tables de routage et des messages de demande et de réponse d'écho que les hôtes utilisent pour déterminer si une destination peut être atteinte.

II.4.1.2. Les problèmes

Dans les systèmes sans connexion, chaque passerelle ou hôte fonctionne d'une manière autonome, en routant et en livrant les datagrammes qu'ils reçoivent sans aucune coordination avec l'émetteur. Le système fonctionne bien quand toutes les machines fonctionnent correctement et s'accordent sur le routage, mais aucun système ne fonctionne correctement d'une façon continue. Des pannes sur les lignes de communication et des processeurs peuvent se produire, Internet ne peut plus livrer les paquets quand la machine de destination est temporairement ou définitivement déconnectée du réseau ou quand le compteur "time-to-live" expire ou quand les passerelles intermédiaires deviennent trop encombrées et ne peuvent plus traiter le trafic.

Les messages ICMP traversent le réseau Internet dans la zone de donnée des datagrammes IP comme tous les autres trafics. Quand un message d'erreurs ICMP arrive à une destination donnée, il est pris en charge par le module IP; il ne passe, donc, pas au programme d'application causant le problème.

En résumé, ICMP permets aux passerelles et aux hôtes de transmettre des messages d'erreurs et des messages de contrôle à d'autres passerelles ou hôtes; ICMP fournit ainsi une communication entre les entités Internet de deux machines en communication.

II.4.2. structure statique

II.4.2.1. Le système de livraison

Chaque message ICMP traverse le réseau Internet dans la partie donnée du datagramme IP. Les datagrammes qui transportent les messages ICMP sont routés exactement comme ceux transportant des informations. Ainsi, même les messages d'erreurs peuvent être perdus ou jetés.

Il est important de savoir que même si les messages ICMP sont introduits et transmis en utilisant IP, ICMP n'est pas considéré comme un protocole de haut niveau, mais une partie nécessaire à IP. La raison pour laquelle on utilise IP pour livrer les messages ICMP est qu'ils peuvent traverser plusieurs réseaux physiques pour atteindre leur destination finale.

II.4.2.2. le format du message ICMP

Chaque message ICMP possède un format propre, ils commencent tous par 3 champs: un champ "TYPE" du message sous forme d'entier à 8 bits, un champ "CODE" également à 8 bits qui fournit des informations complémentaires sur le type de message, et un champ "CHECKSUM" à 16 bits (ICMP utilise le même algorithme de la somme de contrôle que IP, mais cette somme de contrôle d'ICMP se compose seulement du message ICMP).

Les messages ICMP se composent toujours de l'en-tête et les premiers 64 bits de données du datagramme qui a causé l'erreur. La raison pour laquelle on donne toutes ces informations (en-tête du datagramme + 64 premiers bits des données) est de permettre au récepteur de déterminer de façon plus précise les protocoles utilisés et de localiser le programme d'application responsable. La majorité des protocoles de haut niveau sont conçus de façon à ce que les informations cruciales soient encodées dans les premiers 64 bits. Le champ "TYPE" d'ICMP définit la signification du message et le format du reste du paquet. Ces types sont:

Champ type	Type de message ICMP
0	Réponse Echo
3	Destination ne peut être atteinte
4	*Source Quench*
5	Redirection (changer de route)
8	Demande Echo
11	Temp dépassé pour un datagramme
12	Problème de paramètres dans le datagramme
13	Demande d'indication de temps
14	Réponse d'indication de temps
15	Demande d'information
16	Réponse d'information
17	Demande du masque de l'adresse
18	Réponse du masque de l'adresse

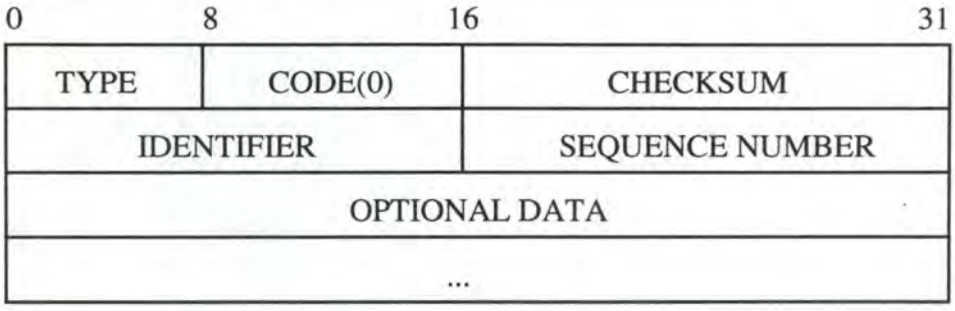
II.4.3. structure dynamique

II.4.3.1. Les tests d'atteinte d'une destination

Un hôte ou une passerelle envoie un message ICMP de demande d'écho pour tester si une destination est vivante et si on pourrait l'atteindre.

Chaque machine qui reçoit cette requête doit formuler une réponse d'écho et la renvoie à la machine qui a envoyé la demande. Le format du message écho est illustré dans la figure ci-dessous:

Le format d'une demande ou réponse d'écho ICMP:

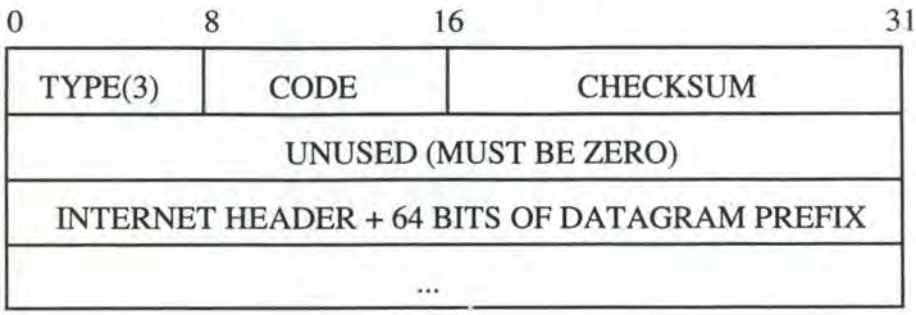


Le champ "OPTIONAL DATA" est un champ de longueur variable, il contient les données qui doivent être retournées au client. Une réponse d'écho doit retourner exactement la même donnée reçue dans la demande. Les champs "IDENTIFIER" et "SEQUENCE NUMBER" sont utilisés par le client pour faire la correspondance entre les demandes et les réponses. La valeur du champ "TYPE" indique si le message est une demande (8) ou une réponse (0).

Le mécanisme demande/réponse d'écho ICMP est un outil très utile dans Internet. Plusieurs systèmes offrent un programme d'application que l'utilisateur peut évoquer pour envoyer des demandes d'écho ICMP et des temps de réponse. Comme le logiciel Internet de la machine de destination manipule ICMP, il est possible d'obtenir un écho même si la machine est très chargée, c'est-à-dire même quand les processus de niveau utilisateur ne réagissent plus.

Quand une passerelle ne peut pas livrer un datagramme IP, elle retourne un message à la source pour lui indiquer que la destination ne peut pas être atteinte, en utilisant le format illustré dans la figure ci-dessous:

Le format du message ICMP indiquant que la destination ne peut être atteinte:



Le champ "CODE" dans le message contient un entier qui donne une description supplémentaire du problème. Les valeurs possibles de ce champ sont:

Valeur du code	Signification
0	On ne peut pas atteindre le réseau
1	On ne peut pas atteindre le hôte
2	On ne peut pas atteindre le protocole
3	On ne peut pas atteindre l'accès
4	Fragmentation nécessaire
5	Route de la source défaillante

Une passerelle envoie des messages indiquant que le réseau ou le hôtes ne peut pas être atteint quand elle ne peut pas router ou livrer des datagrammes. On ne peut pas atteindre des destinations si le matériel est temporairement en panne, ou si l'émetteur spécifie une adresse de destination inexistante, ou (dans des cas rares) si la passerelle n'a pas une route pour un réseau de destination.

Dans le message ICMP, le préfixe du datagramme qui a causé le problème est indiqué pour que le logiciel du protocole à la source peut le connaître avec précision.

Si une passerelle doit fragmenter un datagramme mais le bit de non fragmentation est mis à 1, elle envoie un message de fragmentation nécessaire à la source.

II.4.3.2. Le contrôle de flux

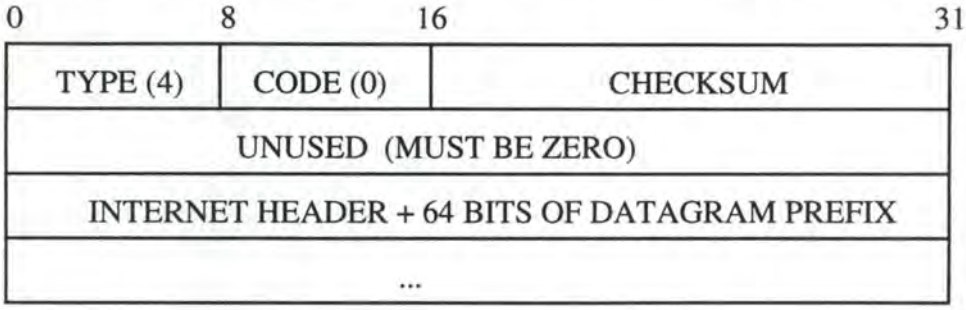
Quand les datagrammes arrivent trop rapidement à une passerelle ou à un hôte pour le traitement, ils sont généralement écartés. Les machines qui écartent ces datagrammes envoient un message ICMP à la source pour lui demander de diminuer son taux de transmission des datagrammes.

D'habitude, les machines envoient une demande à ces sources pour chaque datagramme devant être écarté. Cependant, les passerelles peuvent utiliser plusieurs algorithmes pour sélectionner ces sources.

Pour éviter l'écartement des datagrammes, les passerelles envoient ces demandes de réduction du flux seulement quand leur file d'attente devienne très longue. Il n'y a pas de message ICMP pour inverser l'effet d'une demande de réduction de flux de datagrammes. Au lieu de cela, un hôte qui reçoit des messages de ce type d'une destination *D*, diminue le taux de transmission des datagrammes à *D* jusqu'à ce qu'il ne reçoit plus des demandes; il augmente, ensuite, progressivement le taux tant qu'il n'a pas reçu de nouvelles demandes.

Ces messages contiennent, en plus des champs "TYPE", "CODE" et "CHECKSUM", un champ "UNUSED" de 32 bits, et un champ "DATAGRAM PREFIX" comme illustré dans la figure ci-dessous:

Le format du message ICMP de réduction du taux de trasmission:



Dans tous les messages ICMP, un champ contient un préfixe du datagramme qui a déclenché la demande de réduction du taux de transmission de la source.

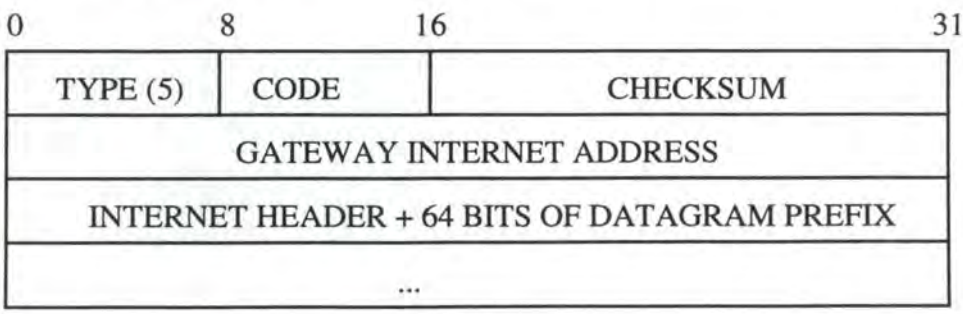
II.4.3.3. Les changements de routes

Les tables de routage d'Internet restent d'habitude statique pendant des périodes de temps assez longues. Les hôtes les initialisent à partir d'un fichier du disque au démarrage du système, et les administrateurs de systèmes changent rarement les routes pendant les opérations normales. Si la structure de l'interconnexion du réseau change, les tables de routage dans des hôtes particuliers peuvent devenir incorrectes. Ces changements peuvent être temporaires ou permanents.

Les passerelles échangent des informations de routage périodiquement pour adapter les changements de réseaux et pour mettre à jour leurs tables de routage. Ainsi, une passerelle connaîssent généralement mieux les routes que les hôtes. Quand une passerelle détecte qu'un hôte est en train d'utiliser une route non optimale, elle lui envoie un message ICMP de redirection. Elle expédie également le datagramme à sa destination.

Chaque message de redirection contient, en plus des champs "TYPE", "CODE", et "CHECKSUM", un champ "GATEWAY INTERNET ADDRESS" de 32 bits et un champ "DATAGRAM PREFIX" comme l'illustre la figure ci-dessous:

Le format du message ICMP de redirection:



Le champ "GATEWAY INTERNET ADDRESS" contient l'adresse d'une passerelle que le hôte doit utiliser pour atteindre la destination mentionnée dans l'en-tête du datagramme. Le champ "INTERNET HEADER" contient l'en-tête IP plus les 64 bits suivants du datagramme qui a causé le message. Ainsi, un hôte qui reçoit un message de redirection examine le préfixe du datagramme pour déterminer l'adresse de destination du datagramme. Le champ "CODE"

d'un message de redirection spécifie davantage comment interpréter l'adresse de destination, en se basant sur les valeurs assignées suivantes:

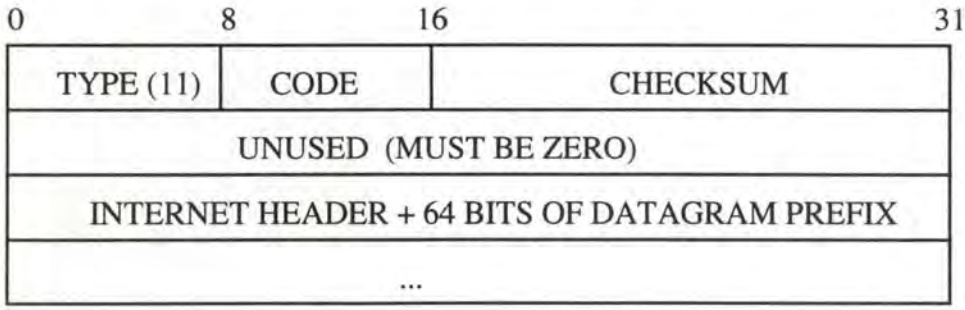
Valeur du code	Signification
0	Rediriger les datagrammes vers le réseau
1	Rediriger les datagrammes vers le hôte
2	Rediriger les datagrammes vers le type de service et le réseau
3	Rediriger les datagrammes vers le type de service et le hôte

Les passerelles peuvent seulement envoyer des demandes de redirection aux hôtes sur des réseaux directement connectés et pas aux autres passerelles.

II.4.3.4. La résolution du problème de routage

Chaque datagramme IP contient un compteur "*time-to-live*", appelé parfois un "*hop count*" . Pour prévenir que les datagrammes tournent en rond indéfiniment, chaque passerelle décrémente le compte et écarte le datagramme quand il atteint zéro. Chaque fois qu'une passerelle écarte un datagramme dont le compte a atteint zéro, elle envoie un message ICMP d'excès de temps à l'émetteur du datagramme, en utilisant le format illustré dans la figure ci-dessous:

Le format du message ICMP de dépassement de temps:



Le champ "CODE" contient une valeur spécifiant la nature du délai d'attente. Les différentes valeurs sont:

Valeur du code	Signification
0	Dépassement du compte *"time-to-live"*
1	Dépassement du délai de réassemblage des fragments

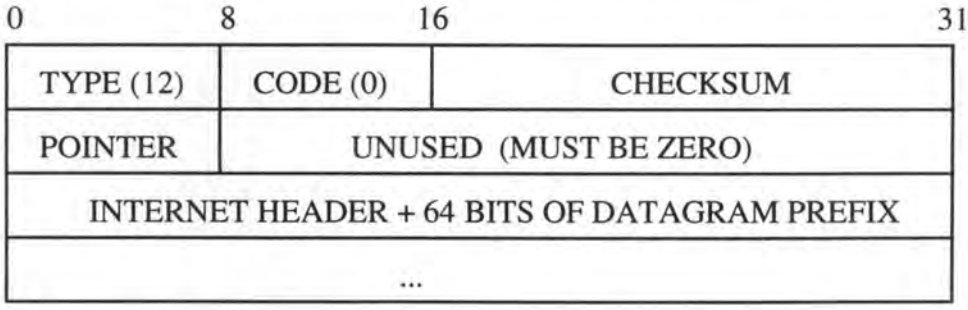
L'assemblage des fragments est une tâche qui consiste à collecter tous les fragments du datagramme. Chaque fois que le premier fragment d'un datagramme arrive, le hôte récepteur déclenche une horloge. Si le délai du temps horloge expire avant l'arrivé de tous les pièces du datagramme, le récepteur le considère comme une erreur. La valeur 1 du champ "CODE" est utilisée pour signaler de telles erreurs à l'émetteur; un message est transmis pour chaque erreur de ce type.

II.4.3.5. La résolution du problème des paramètres d'en-tête

Quand une passerelle ou un hôte rencontre des problèmes avec l'en-tête du datagramme, il envoie un message indiquant un problème de paramètres à l'émetteur.

Une cause possible de tels problèmes se présente quand les arguments d'une option sont incorrects. Le message est transmis seulement quand le problème est tellement sévère que le datagramme doit être écarté. Ce format est illustré dans la figure ci-dessous:

Le format du message ICMP du problème des paramètres:



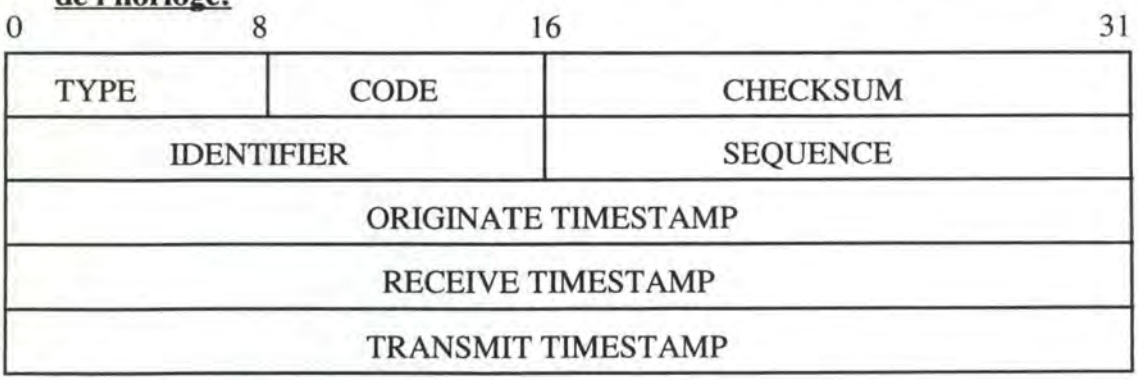
Le champ "POINTER" identifie la position de l'octet, dans l'en-tête du datagramme, qui a causé le problème.

II.6.3.6. La synchronisation de l'horloge et l'estimation du temps de transit

Un client envoie un message "*timestamp request*" à un autre client pour demander à la machine de destination de lui transmettre sa valeur actuelle de l'heure du jour. La machine réceptrice répond en envoyant un "*timestamp reply*" à la machine émettrice.

Le format du message "*timestamp request*" et "*timestamp reply*" sont illustrés dans la figure ci-dessous:

Le format du message ICMP des demandes et réponses de synchronisation de l'horloge:



Les champs "SEQUENCE" et "IDENTIFIER" sont utilisés par la source pour associer les demandes aux réponses. Le champ "TYPE" identifie le message comme une demande (13) ou une réponse (14). Les autres champs spécifient les temps, en millisecondes à partir de minuit.

Le champ "ORIGINATE" est rempli par l'émetteur juste avant la transmission du paquet, le champ "RECEIVER" est rempli immédiatement à la réception d'une demande, et le champ "TRANSMIT" est rempli immédiatement avant qu'une réponse soit transmise.

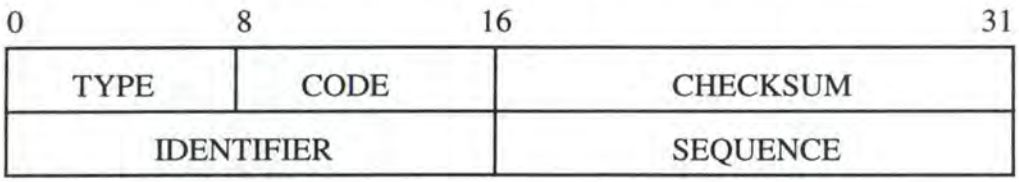
Les hôtes utilisent ces trois champs pour calculer les estimations des délais et pour synchroniser leurs horloges. Comme la réponse comprend le champ "ORIGINATE TIMESTAMP", un hôte peut calculer le durée totale nécessaire pour la transmission d'une demande à une destination, sa transformation en une réponse et son retour.

La réponse transporte à la fois l'heure à laquelle la demande entre dans la machine réceptrice, ainsi que l'heure à laquelle la réponse sort, le hôte peut calculer le temps de transit du réseau, et à partir de cela, estimer les différences des temps horloge des deux machines.

II.4.3.7. L'obtention d'une adresse réseau

Les machines utilisent les messages ICMP de demande d'information pour obtenir une adresse Internet pour un réseau auquel ils sont reliés; c'est une alternative pour RARP. L'émetteur remplie une demande, l'envoie en mettant à zéro l'adresse réseau de destination IP, et attend une réponse. La réponse arrive avec la partie réseau de l'IP de l'émetteur remplie. Les passerelles répondent aux demandes des adresses de réseaux et envoient leurs réponses avec une bonne spécification des deux champs source et destination du datagramme IP. Le format des messages de demande et de réponse est illustré dans la figure ci-dessous:

Le format des messages ICMP de demande et de réponse d'information:



Les champs "IDENTIFIER" et "SEQUENCE" contiennent les valeurs que l'émetteur utilise pour associer des demandes à des réponses. Le champ "TYPE" spécifie si le paquet contient une demande (15) ou une réponse (16).

Chapitre II.5.

LE PROTOCOLE UDP

II.5.1. Objectifs

Le protocole UDP (*User Data Protocol*) est un protocole de transport qui se situe au-dessus d'IP. Il assure un transport non fiable et sans connexion. C'est un protocole qui permet d'envoyer des trames qui contiennent l'adresse IP de la machine émettrice et celle de la machine de destination. Le protocole UDP utilise des numéros de ports pour spécifier une application sur une machine.

UDP est un protocole standard d'Internet, il permet à un programme d'application sur une machine d'envoyer un datagramme à un programme d'application sur une autre machine. Il utilise le protocole Internet pour livrer les datagrammes. D'une façon générale, la seule différence entre UDP et IP est que le message UDP comprends des numéros de port, permettant à l'émetteur de faire une distinction entre plusieurs destinations (programmes d'application) sur la machine cible. UDP comprend également une somme de contrôle de la donnée transmise.

UDP fait ainsi une distinction entre plusieurs processus sur une machine donnée en permettant aux émetteurs et récepteurs d'ajouter à chaque message UDP deux entiers de 16 bits spécifiant les numéros de port. Ces numéros identifient la source et la destination. Quelques numéros de port UDP sont réservés (par exemple: port 69 est réservé pour le protocole Internet "*Trivial File Transfer Protocol*" dit TFTP). D'autres numéros de port sont disponibles pour l'utilisation d'autres programmes d'application.

Dans la structure en couches, UDP est un des protocoles qui se situe au dessus de la couche IP. Ces deux couches sont indépendantes, cependant il y a une forte interaction entre les deux. UDP est un mécanisme qui fournit aux processus utilisateurs un accès au service de livraison de paquets d'Internet qui est non fiable et sans connexion.

II.5.2. Structure statique

II.5.2.1. Les ports

La plupart des ordinateurs modernes sont multitâches, ils permettent l'exécution simultanée de plusieurs programmes d'application. Chaque programme en exécution est un processus (processus niveau utilisateur). Les processus sont créés, puis détruits de façon dynamique. Seuls les émetteurs connaissent les informations nécessaires pour identifier un

processus sur une machine cible. Chaque machine contient un ensemble de points de destination indépendants appelés les ports du protocole (cf. II.1.). Typiquement, chaque port du protocole est identifié par un entier positif. Le système d'exploitation local choisit un mécanisme que les processus utilisent pour spécifier un port ou pour l'accéder.

II.5.2.2. La structure du protocole

Dans le protocole TCP/IP, UDP (pour *User Datagram Protocol*) fourni un mécanisme que l'émetteur utilise pour distinguer une application parmi plusieurs programmes d'application récepteurs sur une même machine. En plus des données transmises par un processus utilisateur, chaque message UDP contient à la fois un numéro de port de destination et un numéro de port de source, permettant ainsi au logiciel UDP de livrer le message au récepteur correspondant, et à ces derniers d'envoyer une réponse.

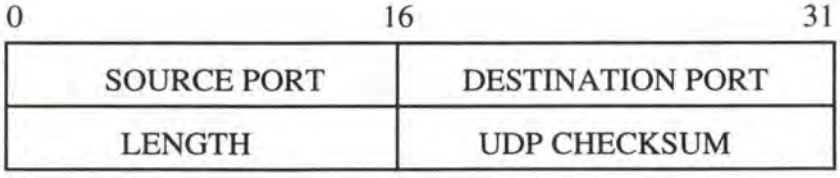
Comme UDP dépend du protocole Internet utilisé pour transporter un message d'une machine à une autre, il fournit le même service qu'IP, c'est-à-dire un service de livraison non fiable et sans connexion. Il n'utilise pas des accusés de réception pour s'assurer de l'arrivée de messages, il n'ordonne pas les messages reçus et il ne fourni pas un "*feedback*" pour le contrôle du taux de transmission des données entre les deux machines. Ainsi, les messages UDP peuvent être perdus, dupliqués ou arrivés en désordre. De plus, les paquets peuvent arriver à la machine de destination à une vitesse supérieure à celle de leur traitement ce qui peut entrainer des risques de surcharge.

UDP fournit, ainsi, un service de livraison non fiable et sans connexion utilisant IP pour le transport des messages entre les machines. Il ajoute des informations pour préciser une destination spécifique sur un ordinateur hôte donné.

II.5.2.3. Le format UDP

Chaque message UDP est appelé un datagramme utilisateur, il se compose de deux parties: l'en-tête et la zone de données. L'en-tête est divisé en 4 champs de 16 bits qui spécifient le port source, le port de destination, la longueur du message et une somme de contrôle. Les détails des champs de l'en-tête sont illustrés dans ce qui suit:

Le format des champs dans l'en-tête UDP:



Les champs "port source" et "port destination" contiennent des numéros de port de 16 bits utilisés pour démultiplexer les datagrammes sur les processus récepteurs. Le port source est optionnel. Il est utilisé pour spécifier le port auquel les réponses doivent être transmises; s'il n'est pas utilisé, sa valeur serait nulle.

Le champ longueur (*length*) contient le nombre d'octets du datagramme, incluant l'en-tête et les données utilisateurs. Ainsi, la valeur minimale de ce champ est égale à 8, la longueur de l'en-tête.

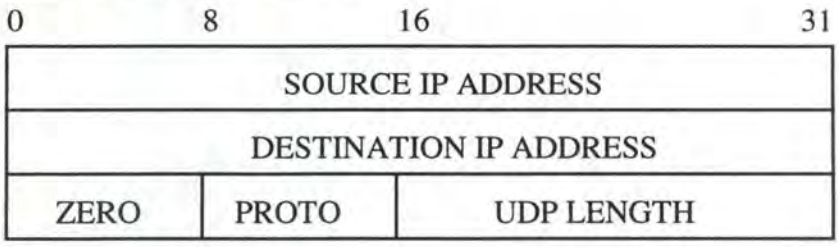
Le champ somme de contrôle UDP est également optionnel; la valeur zéro signifie que la somme n'a pas été calculé. Pour calculer la somme de contrôle, UDP crée un pseudo en-tête, ajoute comme suffixe un octet de zéros pour remplir le datagramme à exactement un multiple de 16 bits, et calcule la somme de contrôle sur la totalité de ce nouveau datagramme. Cet octet de zéros est ajouté à l'en-tête seulement.

L'octet utilisé pour le remplissage et le pseudo en-tête ne sont pas transmis avec le datagramme UDP, ils ne sont pas non plus inclus dans la longueur. UDP utilise le même algorithme de la somme de contrôle qu'IP. Il traite le pseudo en-tête, l'en-tête, la donnée et l'octet de remplissage comme une séquence d'entiers de 16 bits. Pour calculer une somme de contrôle, le logiciel accumule d'abord une somme complémentaire de 16 bits, il prend ensuite les 16 bits complémentaires de la somme pour produire la somme de contrôle. Pendant le calcul de la somme de contrôle, le champ de celui-ci est supposé contenir zéro.

L'en-tête UDP spécifie seulement le numéro de port du protocole. Ainsi, pour vérifier la destination, UDP sur la machine émettrice calcule une somme de contrôle qui couvre l'adresse hôte de destination et le datagramme UDP. A la dernière destination, le logiciel UDP vérifie la somme de contrôle en utilisant l'adresse Internet de destination obtenue de l'en-tête du datagramme IP qui transporte le message UDP. Si la somme de contrôle correspond alors le datagramme a correctement atteint la destination et le port désirés.

Le pseudo en-tête UDP a une longueur de 12 octets arrangée comme suit:

Les octets du pseudo en-tête utilisés pour le calcul de la somme de contrôle:



Les champs adresse source IP et adresse destination IP du pseudo en-tête contiennent les adresses Internet source et destination qui vont être utilisés pour la transmission des messages UDP. Le champ proto contient le type de protocole IP sous forme d'un code, ce code est égal à 17 pour UDP, le champ longueur UDP contient la longueur du datagramme UDP.

Pour vérifier la somme de contrôle, le récepteur doit extraire ces champs de l'en-tête IP, il les assemble en un format pseudo en-tête et ensuite, il recalcule la somme de contrôle après avoir ajouté l'octet de zéros. Si le résultat correspond au champ *udp checksum*, le paquet est intact.

II.5.3. Aspect dynamique

II.5.3.1. UDP et les couches de protocole

La couche UDP se situe, comme on l'a déjà indiqué, au dessus de la couche IP. Les messages UDP complets (en-tête et donnée) sont encapsulés dans la zone de données du datagramme IP avant de traverser l'Internet.

Quand on considère la façon dont les en-têtes sont insérés et enlevés, il est important de penser au principe des couches. En particulier, ce principe s'applique à UDP, ainsi le datagramme UDP reçu dans la couche IP sur la machine de destination est identique au datagramme que UDP passe à IP sur la machine source. Les données qu'UDP livre à un processus utilisateurs sur la machine réceptrice seront exactement les mêmes qu'un processus utilisateur passe à UDP sur la machine émettrice.

La couche IP est seulement responsable des transferts des données entre les hôtes sur Internet alors que la couche UDP est responsable seulement de la différentiation entre les multiples sources ou destinations dans le même hôte. Ainsi, seul l'en-tête IP qui peut identifier les hôtes source et destination; et seule la couche UDP qui peut identifier les ports source ou destination dans un hôte.

II.5.3.2. Le calcule de la somme de contrôle et le modèle en couche

L'adresse source IP dépend de la route choisie pour le datagramme, car l'adresse source identifie l'interface réseau sur laquelle le datagramme est transmis. Ainsi, UDP ne peut pas connaître une adresse source IP à moins qu'il interagisse avec la couche IP.

On suppose que le logiciel UDP demande à la couche IP de calculer les adresses IP, sources et destinations, les utilise pour construire un pseudo en-tête, calcule la somme de contrôle, enlève le pseudo en-tête et passe le datagramme UDP à IP pour la transmission.

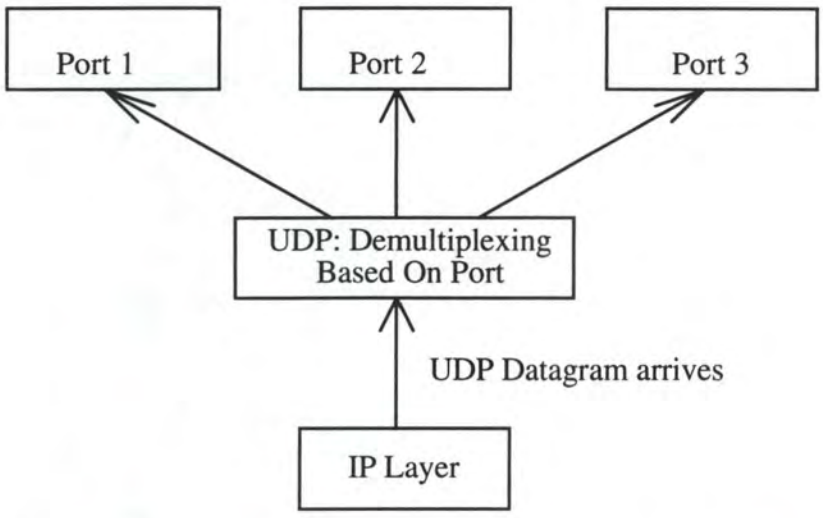
Une autre approche plus efficace peut être définie:
- encapsuler les messages UDP dans les datagrammes IP (dans la couche UDP),
- remplir les champs source et destination IP de l'en-tête,
- calculer la somme de contrôle UDP sur l'en-tête, et
- passer le datagramme IP à la couche IP. Celle-ci va remplir les champs restants de l'en-tête IP.

II.5.3.3. Le démultiplexage UDP

Le terme multiplexage utilisé dans ce modèle est différent de celui du modèle OSI. Il s'agit en fait du routage au niveau de l'adressage interne.

Dans les couches de la hiérarchie du protocole, le logiciel doit multiplexer ou démultiplexer plusieurs choses pour la couche suivante. Le logiciel UDP est un exemple de multiplexage. Il accepte les datagrammes UDP du software IP et les démultiplexe. Un exemple de démultiplexage dans la couche UDP est illustré dans la figure ci-dessous:

Exemple de démultiplexage:



Quand un datagramme UDP arrive, il est routé selon son numéro de port. C'est ce qu'on appelle un démultiplexage basé sur le port.

Les messages UDP utilisent le numéro de port pour sélectionner une destination appropriée des datagrammes reçus.

Chapitre II.6.

DESCRIPTION ET FONCTIONNEMENT DE LA COUCHE TCP

II.6.1. Objectifs et problèmes

II.6.1.1. Les objectifs

Les systèmes de communication des ordinateurs jouent un rôle de plus en plus important dans les environnements militaires, gouvernementals et civils. Comme ces réseaux de communication sont développé et déployé pour des buts stratégiques et tactiques, il est important de fournir des moyens de les interconnecter et de fournir des protocoles de communication standards qui peuvent supporter une large gamme d'applications.

Le protocole TCP (*Transmission Control Protocol*), est considéré comme étant un protocole de communication très fiable entre ordinateurs hôtes dans des réseaux utilisant la commutation de paquets, et dans l'interconnexion de ces réseaux. Il est orienté connexion et adapté aux systèmes de couche de protocoles hiérarchiques. TCP fournit une communication inter-processus fiable entre des paires de processus des ordinateurs hôtes reliés à d'autres réseaux de communication. En principe, TCP devrait être capable d'opérer au-dessus d'un large gamme de systèmes de communication tels que les réseaux à commutation de paquets, à commutation de circuits etc.

Ainsi, le protocole de contrôle de transmission, TCP, définit un service clé offert par l'Internet, à savoir, une livraison fiable des données. Ce service est le plus important et le plus connu des services Internet. TCP offre une connexion en "*Full Duplex*" entre deux machines leur permettant d'échanger des grands volumes de données d'une manière efficace. Comme il utilise un système de fenêtrage, TCP peut également rendre l'utilisation du réseau efficace.

Le service de TCP est flexible, il peut s'adapter à une grande variété de systèmes de transmission supportant IP. Il offre également un système de contrôle de flux permettant la communication entre des systèmes ayant des vitesses variées.

L'unité de base de transfert utilisé par TCP est le segment. Ces segments sont également utilisés pour faire passer les informations de contrôle, par exemple, en permettant au logiciel TCP sur les deux machines d'établir ou d'interrompre une connexion. Le format du segment permet également à une machine d'ajouter (*piggybacking*) dans l'en-tête du segment de donnée circulant dans un sens, un accusé de réception d'une donnée reçue (circulant dans le sens opposé).

TCP implémente le contrôle de flux en informant le récepteur de la quantité de données qu'il pourrait transmettre. Il supporte également des messages hors bande en utilisant un mécanisme de transmission de données urgentes et permet une livraison forcé en utilisant le mécanisme "*push*". Tous ces mécanismes vont être developpés dans les paragraphes qui suivent.

Le protocole TCP est un protocole indépendant malgré qu'il est présenté comme une partie du protocole Internet, il est également un protocole général qui peut être utilisé dans d'autres systèmes. Par exemple, il est possible de l'utiliser dans un réseau du type Ethernet, supportant IP, ou dans des réseaux Internet plus complexes.

TCP offre des spécifications des formats de données et des accusés de réception que peuvent échanger deux ordinateurs pour accomplir un transfert fiable. Il constitue l'ensemble des procédures utilisées par les ordinateurs pour assurer l'arrivé correcte des données. Il spécifie également la manière dont le logiciel TCP distingue un programme d'application cible parmi plusieurs sur une machine donnée, ainsi que la manière par laquelle des machines en communication retrouvent des erreurs telles que les pertes ou les duplications des paquets.

II.6.1.2. Les problèmes de livraison

Les paquets peuvent être perdus ou détruits quand il y a des interférences, quand le matériel tombe en panne ou quand les réseaux deviennent trop chargés. Les réseaux qui routent les paquets peuvent les livrer dans un désordre, ou avec un retard considérable ou peuvent également générer des duplications. De plus, les technologies des réseaux peuvent exiger des tailles optimales des paquets ou posent d'autre contraintes nécessaires pour un transfert efficace.

Un des buts des recherches des protocoles des réseaux était de trouver une solution générale au problème qui consiste à fournir une livraison fiable des données. La construction d'un seul programme de protocole pour que tous les programmes d'application puissent l'utiliser a été envisagée. Un seul protocole peut ainsi contribuer à isoler des programmes d'application les détails du réseau et, par conséquent, rend la définition d'une interface uniforme possible pour le service de transfert de données.

II.6.1.3. Liaison entre interface d'application et service fiable de livraison

L'interface entre des programmes d'application et le service Internet de livraison de données peut prendre cinq caractéristiques:

- **Orientation des données:** Quand deux programmes d'application transfèrent de larges volumes de données, on considère ces données comme des flots de bits, divisés en octets de 8-bits ou bytes. Le service de livraison de données sur la machine de destination passe au récepteur exactement la même séquence d'octets que l'émetteur sur la machine source a transmis.

- **Les connexions:** Le transfert d'un flot de données est analogue à un appel téléphonique. Avant le début du transfert de données, l'émetteur et le récepteur interagissent avec leurs systèmes d'exploitation respectifs. Ceux-ci les informent des demandes de transfert de données.

Généralement, la machine émettrice fait un appel à la machine réceptrice, qui accepte cet appel. Les modules du logiciel du protocole dans les deux systèmes d'exploitations communiquent entre eux en envoyant des messages sur l'Internet vérifiant que le transfert est autorisé, et que les deux machines sont prêtes.
Une fois que ces détails sont réglés, les modules des protocoles informent les programmes d'application qu'une connexion a été établie et que le transfert peut commencer. Pendant le transfert, le logiciel du protocole sur les deux machines vérifie que les données sont reçues correctement tout en continuant la communication. Si la communication échoue, les deux machines détectent la panne et la signale au programme d'application approprié.

On utilise parfois le terme circuit virtuel pour décrire de telle connexion car malgré que des programmes d'application voient la connexion comme un circuit matériel, la fiabilité est une illusion fourni par le service de livraison de données.

- **Transfert avec mémoire-tampon**: Les programmes d'application envoient un flot de données dans le circuit virtuel. Ce transfert se fait byte par byte de façon répétitive. Pour transmettre les données, chaque application utilise les tailles qui lui conviennent. Ces tailles peuvent même être d'un byte. A la réception, le système livre les bytes du flot de données dans l'ordre exacte de leur transmission, et une fois qu'ils sont vérifiés ils seront disponibles au programme d'application récepteur.

  Le logiciel du protocole peut diviser le flot de données en paquets indépendants différentes de ceux qui sont transmis par le programme d'application source. Pour rendre le transfert plus efficace et pour minimiser le trafic du réseau, un certain nombre de données d'un flot sont réassemblés pour remplir un datagramme d'une taille raisonnable avant la transmission. Ainsi, même si le programme d'application génère des flots d'un byte, le transfert dans l'Internet peut être très efficace.

  Pour des applications où les données doivent être livrées même si elles ne remplissent pas la mémoire-tampon, le service de transmission de flots de données offre un mécanisme appelé "*push*" qui est utilisé par les applications pour forcer le transfert. Du côté de l'émetteur, un "*push*" force le logiciel du protocole de transférer toutes les données qui ont été générées sans attendre le remplissage de la mémoire-tampon. Lorsqu'elles atteignent leur destination, le "*push*" va obliger le TCP de mettre les données immédiatement à la disposition de l'application. Un "*push*" garantit le transfert de toutes les données. Ainsi, même si la livraison est forcée, le logiciel du protocole peut choisir de diviser le flot de données à sa façon.

- **Flots de données non structurés:** Le service Internet de livraison de données ne respecte pas les flots de données structurées. Les programmes d'applications utilisant un service de livraison de données doivent bien comprendre le contenu du flot de données et doivent également convenir un format avant de commencer une connexion.

- **Connexion en "Full Duplex":** Les connexions offertes par le service Internet de livraison de données permettent un transfert simultané dans les deux sens. Une telle connexion est appelé "*Full Duplex*". Du point de vue d'un processus d'application, une connexion en "Full Duplex" consiste en deux flots de données indépendants circulant dans les deux sens sans aucune interaction apparente. Ce service de livraison de données permet à un

processus d'application de mettre fin à la circulation d'un flot de données dans un sens, bien que les données continuent à circuler dans l'autre. L'avantage d'une connexion "*Full Duplex*" est que le système peut envoyer des informations de contrôle à la source dans des datagrammes transportant des données. Un tel mécanisme s'appelle le "*piggybacking*", son avantage est qu'il réduit le trafic dans le réseau.

II.6.2. L'aspect statique de la structure

II.6.2.1. Segments, séquences, et numéros de séquences

La séquence d'octets qui doit être transmise est divisée en segments. Chaque segment est envoyé dans un datagramme IP. TCP utilise un mécanisme de fenêtrage pour résoudre deux problèmes importants: l'efficacité de la transmission et le contrôle de flux. Ce mécanisme envoie plusieurs paquets avant de recevoir un accusé de réception. En faisant ainsi, le débit totale augmente puisque le réseau reste occupé.

Le protocole de la technique de fenêtrage résoud également le problème de contrôle de flux de bout en bout, en permettant au récepteur de restreindre la transmission jusqu'à ce qu'il dispose d'un espace suffisant de la mémoire-tampon pour recevoir plus de données.

Le mécanisme de fenêtrage est utilisé au niveau byte, et pas au niveau paquet. Les bytes des données sont numérotés séquentiellement, l'émetteur doit garder trois pointeurs associés à une connexion et qui définissent la fenêtre. Le premier pointeur marque la partie gauche de la fenêtre, séparant les bytes qui ont été transmis et qui ont reçus leurs accusés de réception, des bytes qui vont être transmis. Un second pointeur marque la partie droite de la fenêtre et définit le byte le plus élevé dans la séquence. Ce byte peut être transmis avant que d'autres accusés de réception soient reçus. Le troisième pointeur marque les limites qui séparent les bytes de la fenêtre qui ont déjà été transmis de ceux qui ne le sont pas encore. Un exemple de fenêtrage est illustré dans les figures II.6.3.2.(a) et II.6.3.2.(b).

Le fonctionnement de la fenêtre au niveau de l'émetteur va être développé dans le paragraphe II.6.3. Le récepteur doit également maintenir une fenêtre similaire pour réassembler les données.

Le caractère "*Full Duplex*" de la transmission signifie que deux transferts peuvent s'effectuer simultanément sur chaque connexion, et dans les deux sens. Les transferts sont totalement indépendants car à tout moment les données peuvent circuler dans un ou dans les deux sens. Ainsi, le logiciel de TCP maintient deux fenêtres pour chaque connexion, une est utilisée pour les données à transmettre et l'autre pour les données reçues.

II.6.2.2. La variation de la taille de la fenêtre et le contrôle de flux

TCP permet la variation de la taille des fenêtres au cours du temps. Chaque accusé de réception contient une annonce de fenêtre (*window advertisement*) qui indique le nombre additionnel de bytes de données que le récepteur est prêt à accepter. En réponse à une annonce d'accroissement de la fenêtre, l'émetteur augmente la taille de sa fenêtre et procède à l'émission des octets dont il n'a pas encore reçue les accusés de réception. Si, par contre, il

reçoit une information indiquant un rétrécissement de la fenêtre, l'émetteur réduit la taille de sa fenêtre et ne transmet plus les bytes qui dépassent les limites annoncées. Les annonces accompagnent les accusés de réception, ainsi, la taille de la fenêtre change quand celle-ci se déplace à droite.

L'avantage d'utiliser une taille variable de la fenêtre est de pouvoir contrôler le flux de données ainsi que la fiabilité du transfert. Si l'espace de la mémoire-tampon du récepteur commence à se remplir, il ne peut plus tolérer davantage de paquets, alors il envoie une annonce de petite fenêtre. Dans le cas extrême, le récepteur annonce une taille de fenêtre nulle pour stopper toutes les transmissions. Et quand l'espace de sa mémoire-tampon se vide, le récepteur annonce une taille de fenêtre non nulle pour déclencher de nouveau le flux de données.

Il est important d'avoir un mécanisme de contrôle de flux dans un environnement Internet. Car dans un tel environnement des machines de différentes vitesses et de différentes tailles communiquent sur des réseaux et à travers des passerelles qui ont eux même des vitesses différentes et sont de capacités différentes. Il y a deux problèmes indépendants liés au contrôle de flux:

1. Les protocoles Internet nécessitent un contrôle de flux de bout en bout entre la source et la destination. Quand un mini-ordinateur, par exemple, communique avec un "*mainframe*", le mini-ordinateur doit régler l'afflux de données, pour éviter que le système soit rapidement envahi. Ainsi, TCP doit implémenter un contrôle de flux de bout en bout pour garantir une transmission fiable.

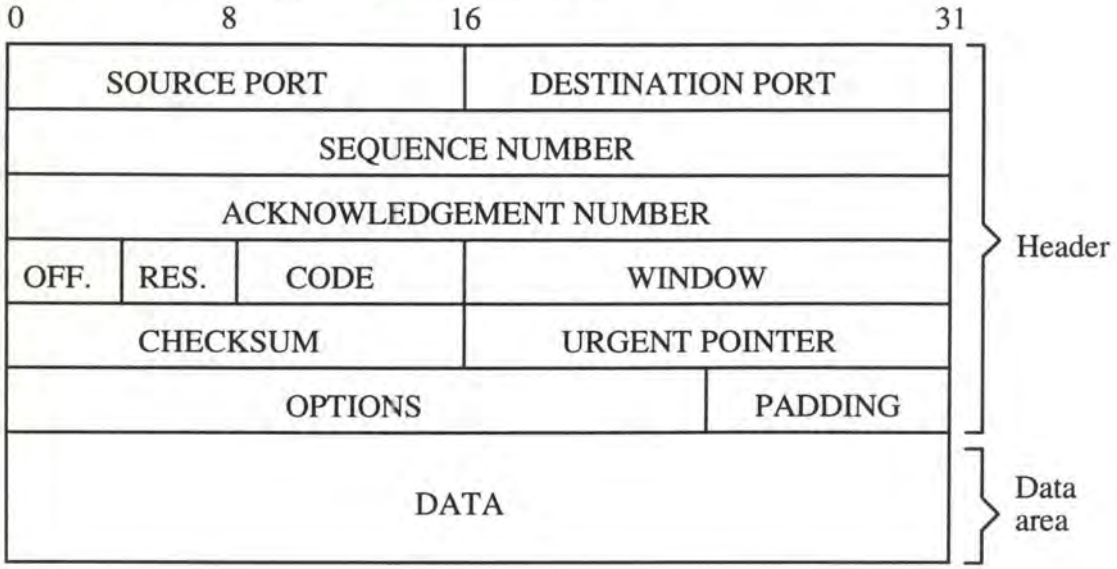
2. Les protocoles Internet nécessitent un mécanisme de contrôle de flux qui permet aux machines intermédiaires, comme les passerelles, de contrôler une source qui transmet plus de trafic qu'ils ne peuvent tolérer.

TCP combine la technique de fenêtrage avec le mécanisme de contrôle de flux pour résoudre le problème de contrôle de flux de bout en bout. Le deuxième problème est résolu par ces machines intermédiaires en utilisant les messages "*source quench*" d'ICMP. Ce protocole a été développé dans un le chapitre II.4.

II.6.2.3. Format du segment TCP

L'unité de transfert entre les modules TCP de deux machines est appelée "segment". Des segments sont échangés pour établir une connexion, transférer des données, transmettre des accusés de réception, annoncer la taille de la fenêtre et pour fermer une connexion. TCP utilise la technique du "*piggybacking*". Cette technique permet la transmission d'un accusé de réception d'une machine A vers une machine B dans le même segment que celui des données. Le format d'un segment TCP est illustré dans figure II.6.2.3. ci-dessous:

Figure II.6.2.3. Le format du segment TCP:



Chaque segment est divisé en deux parties, l'en-tête suivie de la zone de données. L'en-tête est souvent appelé le "*TCP header*". Les champs "SOURCE PORT" et "DESTINATION PORT" de l'en-tête contiennent les numéros d'accès qui identifient les programmes d'application sur les machines connectées.

Les segments TCP sont transmis comme les datagrammes Internet. L'en-tête transporte plusieurs champs d'information. Un en-tête TCP est encapsulé dans la zone de données du paquet IP, il fournit des informations spécifiques au protocole TCP.

Le champ "SEQUENCE NUMBER" identifie la position de la donnée du segment dans la séquence de bytes transmise par l'émetteur. Le champ "ACKNOWLEDGEMENT NUMBER" identifie la position du byte le plus élevé que la source a reçu. On remarque que le numéro de séquence fait référence aux flots de données circulant dans le même sens que le segment, par contre le numéro de l'accusé de réception fait référence aux flots de données circulant dans le sens opposé du segment.

Le champ "OFF." pour "*offset*" contient un entier de 4 bits, il indique l'emplacement des donnée dans le segment. Cet entier est nécessaire car le champ "OPTIONS" varie en longueur, il dépend de l'option choisie. Le champ "RES." est réservé pour une utilisation future.

Certains segments transportent seulement un accusé de réception, d'autres transportent des données. Il y a des segments qui transportent des demandes d'ouverture ou de fermeture de connexion.

Le logiciel de TCP utilise un champ "CODE" de 6 bits pour déterminer le but et le contenu du segment. Les 6 bits indiquent comment interpréter d'autres champs de l'en-tête. Le tableau II.6.2.3. suivant résume les différents codes de contrôle:

Tableau II.6.2.3. Les différents codes de contrôle:

Bit (de gauche à droite)	Explication
URG	Champ valide de pointeur urgent
ACK	Champ valide d'accusé de réception
PSH	Ce segment demande un "push"
RST	Remise à zéro de la connexion
SYN	Numéros de séquences synchronisés
FIN	L'émetteur a atteint la fin de l'émission

Chaque fois que le logiciel de TCP transmet un segment, il annonce le nombre de données qu'il est prêt à accepter en spécifiant la taille de sa mémoire-tampon dans le champ "WINDOW". L'annonce de la taille de la fenêtre est un autre exemple de l'utilisation de la technique du "*piggybacking*" puisqu'elle accompagne tous les segments y compris ceux transportant des données ou seulement des accusés de réception.

TCP permet à l'émetteur de spécifier le caractère urgent de quelques données, c'est-à-dire que les données doivent être transmises dans les plus brefs délais.

Les données urgentes contiennent des messages différents des données normales. Par exemple, un trafic urgent pourrait inclure des signaux d'interruption tapés au clavier. Un tel trafic est souvent appelé "*out of band traffic*".

Le détail exact de la façon dont TCP informe le programme d'application de l'urgence de la donnée dépend du système d'exploitation de l'ordinateur. Le mécanisme utilisé pour marquer les données urgentes utilise les deux champs: "URG" et "URGENT POINTER". Quand le bit URG est mis à 1, le pointeur urgent spécifie une position dans la séquence de bytes où se terminent les données urgentes.

Le logiciel de TCP utilise le champ "OPTIONS" pour communiquer avec la machine réceptrice. En particulier, un récepteur peut spécifier la taille maximale du segment qu'il désire recevoir. Il est donc important de permettre au récepteur de spécifier une taille maximale d'un segment surtout si un petit ordinateur reçoit des données d'un plus grand ordinateur.

Il est difficile de choisir la bonne taille maximale du segment car les performances peuvent être faibles pour des tailles de segments très larges ou très petites. Par conséquence, quand la taille du segment diminue, l'utilisation du réseau diminue également. Ceci est dû au fait que les segments TCP qui circulent dans le réseau sont encapsulés dans des datagrammes IP, qui sont eux même encapsulés dans des trames physiques du réseau.

Ainsi, chaque segment a au moins 40 octets d'en-tête (TCP et IP) en plus de la donnée, et les datagrammes transportant un octet de données utilisent au plus 1/40 de la largeur de bande du réseau pour les données utilisateurs. D'autre part, des tailles de segments extrêmement larges produisent également des faibles performances. Les segments larges aboutissent à de larges datagrammes qui sont fragmentés avant la transmission. Les fragments, contrairement aux datagrammes, ne sont pas indépendants; tous les fragments doivent arriver ou doivent tous

être retransmis. Si la probabilité de la perte d'un paquet est non nulle, l'augmentation de la taille du segment au dessus du seuil de fragmentation diminue le débit.

II.6.3. L'aspect dynamique

II.6.3.1. La fiabilité

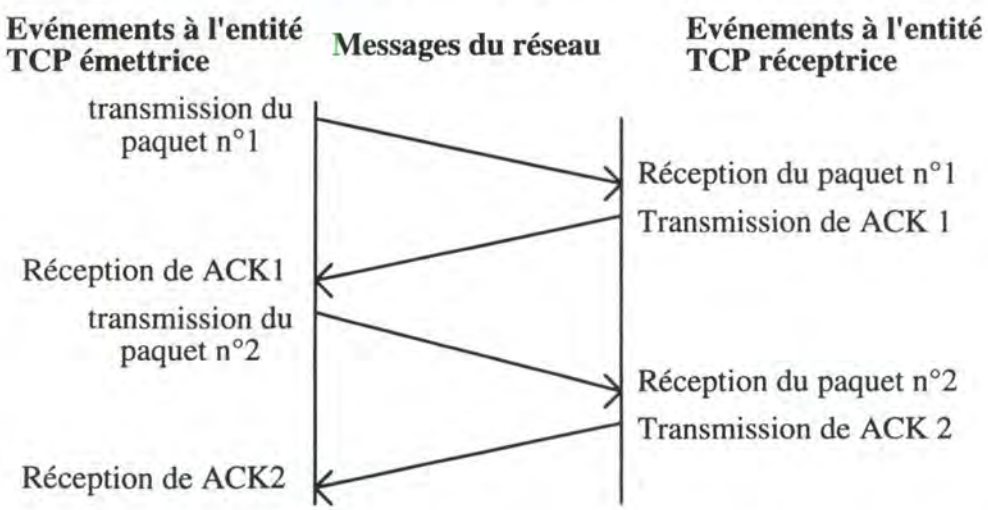
Le service TCP garantit une livraison sans duplication et sans perte de données entre deux machines. Pour ce faire, TCP utilise une seule technique fondamentale connu sous le nom "*positive acknowledgement with retransmission*".

Cette technique exige du récepteur de communiquer avec la source, en lui envoyant un accusé de réception chaque fois qu'il reçoit des données. L'émetteur garde une trace de chaque paquet qu'il transmet et attend un accusé de réception avant de transmettre le prochain paquet. Il déclenche également une horloge quand il envoie un paquet et retransmet ce même paquet si le délai expire avant l'arrivé de son accusé de réception.

Cet échange est illustré dans la figure II.6.3.1.(a) ci-dessous. les événements dans l'entité TCP émettrice et réceptrice se trouvent à gauche et à droite. Les lignes diagonales qui traversent le milieu montrent le passage d'un message dans le réseau.

La deuxième figure II.6.3.1.(b) utilise le même format de diagramme et montre ce qui se passe quand un paquet est perdu ou corrompu. L'émetteur déclenche une horloge après la transmission d'un paquet, quand le délai expire, il suppose que le paquet est perdu et le retransmet.

Figure II.6.3.1.(a) Un simple tranfert de données avec accusés de réception:



Le protocole utilise la technique "*positive acknowledgement with retransmission*" avec laquelle l'émetteur attend un accusé de réception (ACK) pour chaque paquet transmis; mais ceci n'implique pas qu'il faut attendre l'ACK avant d'envoyer le paquet suivant.

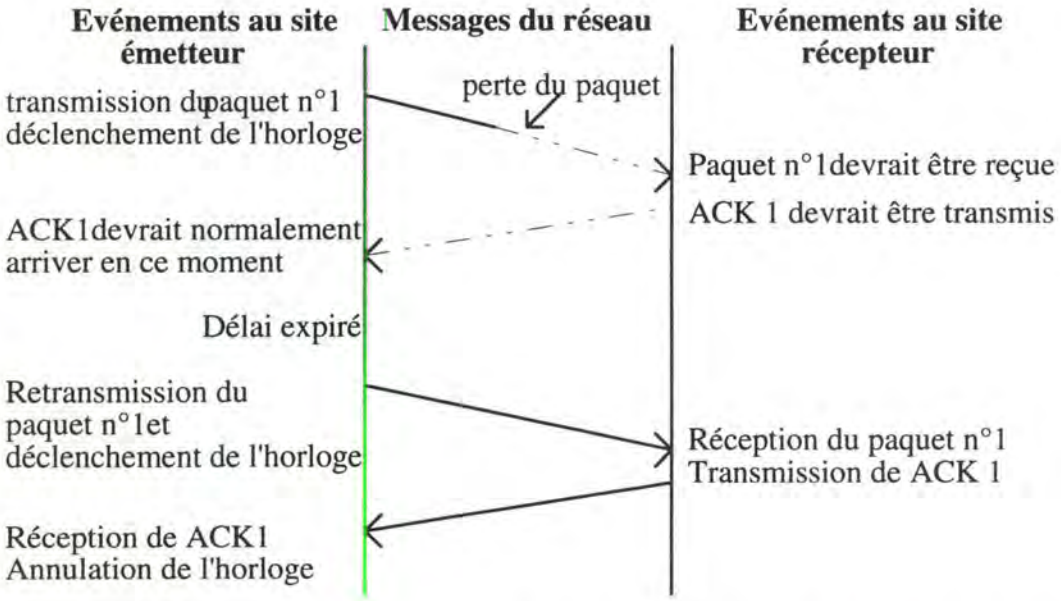
Les lignes verticales représentent le déroulement chronologique des événements dans chaque site.

Un autre problème de fiabilité apparaît quand un système de livraison duplique les paquets. Cette duplication peut également se présenter quand il y a des grands retards sur les réseaux, ceci implique des retransmissions prématurées. Il faut être très prudent pour résoudre ce problème car on peut avoir une duplication aussi bien des paquets que des accusés de réception..

Les protocoles de fiabilité détectent habituellement la duplication des paquets en assignant à chaque paquet un nombre de séquence et en exigeant du récepteur de se rappeler de ce nombre qu'il a reçu.

Pour éviter la confusion causé par les accusés de réception des paquets retardés ou dupliqués, les "*positive acknowledgement protocols*" envoient des nombres de séquence en retour dans l'accusé de réception, ainsi le récepteur peut correctement associer ces accusés aux paquets transmis.

Figure II.6.3.1.(b) Un tranfert avec perte de données:



Le dépassement du délai et la retransmission qui se produit quand un paquet est perdu, sont schématisé dans la figure ci dessus. Les lignes en pointillé indiquent le temps nécessaire pour la transmission d'un paquet et de son accusé de réception, si le paquet n'était pas perdu.

II.6.3.2. La technique de fenêtrage

Pour rendre la transmission des flots de données efficace, on utilise la technique de fenêtre à glissière ou de "fenêtrage". L'émetteur, après avoir transmis un paquet, attend un accusé de réception avant de transmettre un autre.

Comme la figure (a) ci-dessus le montre, à tout moment, les données circulent entre les machines dans une seul sens, même si le réseau a une capacité de communication simultanée et bidirectionnelle (*Full Duplex*). Le réseau serait souvent en repos si les machines tardent pour répondre. Par exemple, pendant que les machines calculent les routes ou les sommes de contrôle.

Si on dispose d'un réseau ayant des retards de transmissions assez longs, le problème devient plus claire: un "*simple positive acknowledgement protocol*" gaspille une grande quantité de la capacité du réseau car il doit retarder l'émission d'un nouveau paquet jusqu'à ce qu'il reçoit l'accusé de réception du dernier paquet transmis.

La technique de fenêtrage est une forme plus complexe que la technique du "*positive acknowledgement and retransmission*". Les protocoles de fenêtrage utilisent mieux la largeur de bande du réseau car ils permettent à l'émetteur de transmettre plusieurs paquets sans attendre un accusé de réception. Supposons qu'on dispose d'une séquence de paquets qu'on va transmettre. Le protocole va placer une petite fenêtre sur la séquence et transmet tous les paquets qui se trouvent à l'intérieur de cette fenêtre. Dans la figure II.6.3.2.(a) ci-dessous on a un protocole de fenêtrage à huit paquets et dans la figure II.6.3.2.(b) la fenêtre glisse de telle façon que le paquet 9 puisse être transmis et ceci après avoir reçu un accusé de réception du paquet 1.

Figure II.6.3.2.(a) Fenêtre initiale:

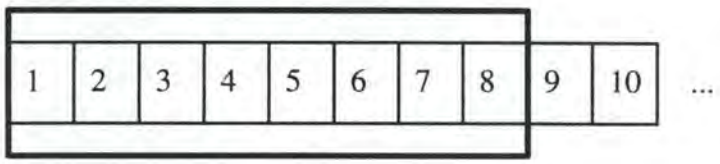
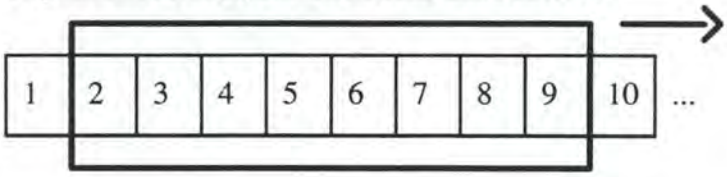


Figure II.6.3.2.(b) Glissement de la fenêtre:



Les fenêtres de l'exemple ci-dessus ont la taille 8, l'émetteur peut transmettre 8 paquets avant de recevoir un accusé de réception. Une fois que l'émetteur reçoit l'accusé de réception du premier paquet, il fait glisser la fenêtre d'un paquet vers la droite, comme le montre la figure ci-dessus, et transmet le paquet suivant. La fenêtre continue à glisser tant que des accusés de réception sont reçus. La performance de ce protocole dépend de la taille et la vitesse à laquelle le réseau reçoit les paquets.

Le concept clé est que l'émetteur peut transmettre tous les paquets qui se trouvent à l'intérieur de la fenêtre sans attendre un accusé de réception. Ainsi, en augmentant la taille de la fenêtre, il est possible d'éliminer complètement le temps perdu par le réseau. Et donc avec un protocole de fenêtrage optimale, le réseau sera complètement saturé avec des paquets, et on obtient un plus grand débit qu'un simple protocole de fenêtrage de taille 1.

Généralement, ce protocole doit toujours se rappeler des paquets qui ont reçus leurs accusés de réception et tient pour chaque paquets, n'ayant pas encore reçu un accusé, un "*timer*". Si le paquet est perdu, le délai sera expiré et l'émetteur retransmet ce paquet. Quand l'émetteur glisse la fenêtre, il dépasse tous les paquets dont l'accusé de réception est déjà reçu.

Du côté du récepteur, le système garde une fenêtre analogue, pour recevoir et accuser réception des paquets qui arrivent. Ainsi, la fenêtre partitionne la séquence de paquets en trois groupes: celui qui se trouve à gauche de la fenêtre contient les paquets qui sont déjà transmis

avec succès, reçus et on a eu leurs accusés de réception; ceux qui sont à droite ne sont pas encore transmis et ceux qui se trouvent à l'intérieur de la fenêtre sont en cours de transmission. Le paquet ayant le numéro le plus bas dans la fenêtre est le premier paquet déjà transmis de la séquence dont l'accusé de réception n'est pas encore reçu.

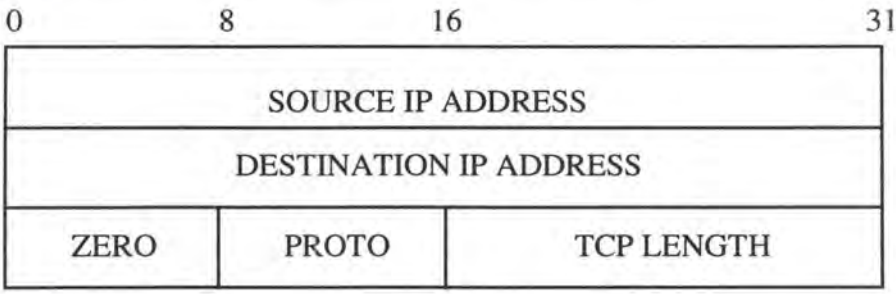
II.6.3.3. Le calcul de la somme de contrôle

Le champ "CHECKSUM" dans l'en-tête du segment TCP contient un entier de 16 bits utilisé pour vérifier l'intégrité de l'en-tête du segment et de la donnée. Pour calculer cette somme de contrôle, TCP sur la machine émettrice prépare un pseudo en-tête du segment, ajoute plusieurs bytes de valeur nulle pour remplir le segment complet (*header* + *data*) afin d'arriver à un multiple de 16 bits, et calcule la somme de contrôle de tout le résultat.

TCP ne compte pas les zéros ajoutés dans la longueur et ne les transmet pas. Il suppose également que le champ "CHECKSUM" vaut zéro pour faciliter le calcul de la valeur de contrôle. Comme pour d'autres valeurs de contrôle, TCP utilise une arithmétique à 16 bits. Au site récepteur, le logiciel de TCP exécute le même calcul pour vérifier si le segment reçu est intacte.

Le but de l'utilisation d'un pseudo en-tête est exactement le même que dans UDP (User Datagram Protocol) (voir II.5.). Il permet au récepteur de vérifier que le segment a atteint correctement sa destination, celle-ci se compose de l'adresse Internet du hôte et du numéro d'accès du protocole. La figure II.6.3.3. ci-dessous illustre le format du pseudo en-tête utilisé dans le calcul de la somme de contrôle:

Figure II.6.3.3. Format du "pseudo header":



Dans le "*pseudo header*", le champ "PROTO" spécifie le type du protocole utilisé, et le champ "TCP LENGTH" spécifie la longueur totale du segment TCP.

Le récepteur extrait l'information du datagramme IP qui transporte le segment, analyse tous ses champs, recalcule la somme de contrôle pour la vérification et enfin enlève les zéros ajoutés.

II.6.3.4. Accusé de réception et retransmission

Comme TCP transmet les données dans des segments de longueurs variables, les accusés de réception s'appliquent à une position dans la séquence, et pas aux paquets ou aux segments. Chaque accusé de réception spécifie un byte plus grand que la plus haute position de byte qui a été reçu.

L'émetteur reçoit des "*feed-back*" continus du récepteur chaque fois qu'il progresse dans la séquence. Ainsi, les accusés de réception spécifient toujours le numéro du prochain byte que le récepteur s'attend à recevoir. Le mécanisme des accusés de réception TCP est appelé "*cumulative*" car il donne le nombre de séquences qui ont été accumulées. Ces accusés de réception cumulatifs ont à la fois des avantages et des inconvénients.

Un des avantages est que les accusés de réception sont à la fois facile à générer et non ambigus. Un autre avantage est que les accusés de réception perdus ne forcent pas nécessairement la retransmission. L'émetteur ne reçoit pas des informations concernant toutes les transmissions réussites, mais seulement une seule position dans la séquence qui a été reçue.

II.6.3.5. Le délai d'attente et la retransmission

TCP s'attend à recevoir des accusés de réception de la destination chaque fois que ce dernier reçoit des données. Chaque fois qu'il envoie un segment, TCP déclenche une horloge et attend un accusé de réception. Si le délai est expiré avant l'arrivée de l'accusé de réception du segment de données, TCP suppose que le segment est perdu ou corrompu et le retransmet.

L'algorithme de retransmission TCP diffère des autres algorithmes utilisés dans plusieurs protocoles de réseaux. TCP est destiné à des environnements Internet où le chemin entre deux machines peut traverser un seul réseau à grande vitesse, où plusieurs passerelles à travers plusieurs réseaux intermédiaires. Il est donc impossible de connaître à priori à quelle vitesse les accusés de réception arrivent à la source. Le retard mis dépend du trafic, et le temps de transmission d'un segment et la réception de son accusé de réception sera variable.

TCP reçoit les différents délais Internet, et utilise un algorithme adaptatif avec ces délais. Pour collecter les données nécessaires pour l'algorithme adaptatif, TCP enregistre le moment auquel chaque segment est transmis, et le moment d'arrivé de l'accusé de réception des données de ce segment. Avec ces deux moments, TCP calcule le temps écoulé connu sous le nom de "*round trip time*". Chaque fois qu'il dispose d'un nouveau "*round trip time*", TCP ajuste sa moyenne de durée de transfert.

Généralement, le logiciel de TCP garde une durée de transfert moyenne comme une référence et utilise les nouvelles durées pour modifier lentement cette référence. En bref, TCP collecte les différents intervalles de temps rencontrés dans l'environnement Internet, il utilise un algorithme de retransmission adaptatif qui contrôle ces délais et ajuste son paramètre du délai d'attente.

II.6.3.6. La réponse à l'encombrement

TCP considére l'interaction entre les deux extrémités d'un connexion et les délais de communication entre eux. En pratique, cependant, TCP doit également réagir à l'encombrement de l'Internet. L'encombrement a pour conséquence des retards sévères causés par une surcharge de datagrammes dans un ou plusieurs points de commutation (dans les passerelles par exemple).

Quand il y a encombrement, les retards augmentent et la file d'attente des datagrammes devant les passerelles s'agrandit. Chaque passerelle possède une capacité de chargement limitée, il y a donc une concurrence pour les datagrammes. Dans le pire des cas, le nombre total de datagrammes qui arrivent à une passerelle encombrée augmente jusqu'à ce que la passerelle atteint sa capacité maximale et commence à écarter des datagrammes.

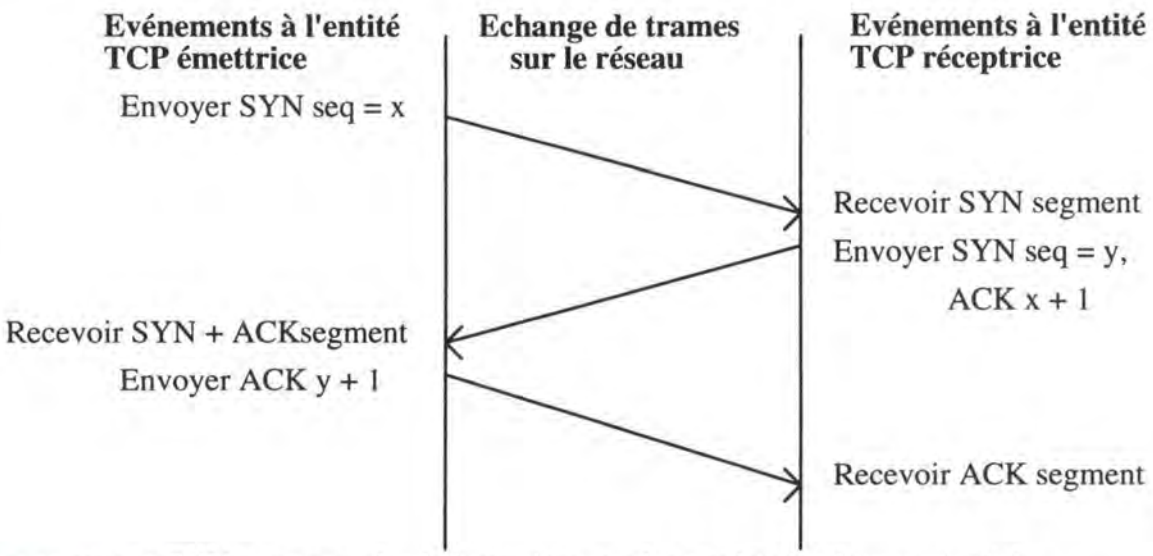
Quand il y a un encombrement, TCP répond en réduisant le taux de transmission. Les passerelles peuvent utiliser des techniques telle que ICMP (*Internet Control Message Protocol*) (voir II.4.) pour informer les hôtes qu'un encombrement s'est produit. Mais les protocoles de niveau transport peuvent détecter l'encombrement automatiquement en observant l'augmentation des durées des délais de transfert.

Si la conception de TCP était pauvre, l'encombrement serait aggravé puisque l'augmentation des délais et les fréquentes retransmissions de segments seraient ignorés. Cependant, un TCP bien conçu aide à alléger l'encombrement.

II.6.3.7. L'établissement d'une connexion

Pour établir une connexion, TCP utilise la technique du "*three-way handshake*". Dans le cas le plus simple, cette technique procède comme il est illustré dans la figure II.6.3.7. ci-dessous (Comer 88):

Figure II.6.3.7. La séquence de messages dans un "three-way handshake":



Les segments SYN transporte les informations sur le numéro de séquence initial.
La technique du "*three-way handshake*" accomplit deux fonctions importantes. Elle garantit que les deux extrémités sont prêtes à transférer les données (et elles savent qu'elles sont toutes les deux prêtes). Elle leur permet de s'accorder sur les numéros initiaux des séquences. Les numéros de séquences sont transmis et reconnus pendant le "*Three-way handshake*". Chaque machine choisit un numéro de séquence initial qui va être utilisé pour identifier les bytes dans le flot de données qu'elle va transmettre.

Il est important que les deux extrémités s'accordent sur le numéro initial pour que les numéros de bytes utilisés dans les accusés de réception s'accordent avec ceux utilisés dans les segments

de données. Les machines s'accordent sur les numéros de séquences pour deux flots de données après seulement trois messages, puisque chaque segment contient à la fois un champ du numéro de séquence et un champ de l'accusé de réception.

Le premier segment d'un "*Three-way handshake*" peut être identifié car il a le bit SYN dans le champ code à 1. Le deuxième message a à la fois le bit SYN et les bits ACK, indiquant qu'il accuse réception du premier segment SYN et qu'il continue le "*handshake*". Le dernier message est seulement un accusé de réception et est simplement utilisé pour informer la destination que les deux côtés sont d'accord sur l'établissement de la connexion. Généralement, le logiciel de TCP sur une machine attend passivement le "*Three-way handshake*", sur l'autre machine il l'amorce.

Le "*Three-way handshake*" est bien conçu pour fonctionner même si les deux machines tentent d'établir une connexion simultanément. Une fois la connexion établie, les données peuvent circuler dans les deux sens. Comme les messages transmis peuvent se perdre ou arriver en retard, le protocole doit utiliser les mécanismes de délais d'attente et de retransmission des requêtes perdues.

D'autres problèmes peuvent, également, apparaître quand les demandes de retransmission arrivent pendant l'établissement de la connexion, ou quand elles sont retardées jusqu'à ce qu'une connexion soit établie, utilisée et terminée. La technique du "*three-way handshake*" résout ces problèmes.

La machine *A*, qui se trouve dans le site 1, qui amorce le "*handshake*", passe son numéro de séquence initial x, dans le premier champ de séquence SYN du premier segment du "*three-way handshake*". La deuxième machine *B*, qui se trouve dans le site 2, reçoit le SYN, enregistre le numéro de séquence et répond en envoyant son numéro de séquence initial dans le champ séquence ainsi qu'un accusé de réception qui spécifie que *B* attend le byte x + 1.

Dans le dernier message du "*handshake*", *A* reconnaît tous les bytes jusqu'au numéro y qu'elle reçoit de *B*. Dans tous les cas, des accusés de réception suivent la convention en utilisant le prochain numéro de byte attendu. Avec une telle conception du protocole, il est possible de transmettre des données avec les numéros de séquence initiaux dans les segments du "*handshake*". Pour cela, le logiciel du protocole doit garder les données jusqu'à la terminaison du "*handshake*". Une fois la connexion est établie, le logiciel de TCP peut libérer ces données et les livrer rapidement au programme d'application qui les attend.

II.6.3.8. La fermeture d'une connexion

Les connexions TCP sont "*Full Duplex*" et donc elles peuvent être vues comme si elles contenaient deux transferts indépendants de données, dans les deux sens. Quand un programme d'application dit à TCP qu'il n'a plus de données à transférer, TCP ferme la connexion dans ce sens. Pour fermer l'autre moitié de la connexion, l'émetteur termine la transmission des données restantes et envoie un segment avec le bit FIN mis à 1.

Le TCP récepteur accuse réception du segment FIN et informe le programme d'application qu'il n'y a plus de données disponibles. Une fois la connexion fermée dans un des deux sens, TCP refuse d'accepter des données émises dans ce sens. Pendant ce temps, les données peuvent continuer à circuler dans le sens opposé jusqu'à sa fermeture par l'émetteur. Quand les

connexions sont fermées dans les deux sens, les données ne circulent plus et la connexion est supprimée.

II.6.3.9. Remise à zéro de la connexion

Un programme d'application ferme une connexion quand il finit de l'utiliser. Ainsi, la fermeture est considérée comme une utilisation normale. Parfois, des conditions anormales apparaissent et force un programme d'application ou le logiciel du réseau de rompre la connexion. TCP offre une opération de remise à zéro pour de telles déconnexions anormales.

Une extrémité de la connexion amorce une remise à zéro en envoyant un segment avec le bit RST dans le champ CODE mis à 1. L'autre extrémité répond immédiatement à une remise à zéro en faisant échouer la connexion. Elle informe également le programme d'application qu'une remise à zéro vient de se produire. Un avortement signifie que les transferts dans les deux sens cessent immédiatement, et les ressources telles que les mémoires-tampon seront libérées.

II.6.3.10. Livraison forcée

TCP peut diviser le flot de données en segments pour les transmettre sans tenir compte des tailles de transfert utilisées par les programmes d'application. L'avantage principal, permettant à TCP de choisir une division, est l'efficacité. Il peut accumuler plusieurs bytes dans une mémoire-tampon pour rendre les segments essez longs, réduisant les frais qui se présentent quand les segments contiennent seulement peu de bytes de données.

Malgré que la mémoire-tampon améliore le débit du réseau, elle peut déranger quelques applications. Si on utilise une connexion TCP pour passer des caractères entre un terminal interactif et une machine distante, l'utilisateur s'attendrait à une réponse instantanée après chaque frappe au clavier. Si TCP met les données dans ses mémoires-tampon, la réponse pourrait être retardée.

TCP fournit une opération "*push*" qui peut être utilisée par un programme d'application pour forcer la livraison des données actuellement dans le flot de données sans attendre que la mémoire-tampon les remplisse. L'opération "*push*" fait plus que forcer TCP de transmettre un segment. Elle demande également à TCP de mettre à 1 le bit PSH dans le champ code du segment, de façon à ce que les données seront livrées au programme d'application récepteur.

Ainsi, quand on envoie des données d'un terminal interactif, l'application utilise la fonction "*push*" après chaque frappe au clavier. De façon similaire, les programmes d'application peuvent envoyer et éditer rapidement la sortie standard sur le terminal en faisant appel à la fonction "*push*" après avoir écrit un caractère ou une ligne.

En plus de la fonction "*push*", TCP fournit un mécanisme appelé pointeur urgent qui permet à l'émetteur d'informer le récepteur qu'une donnée urgente va arriver et doit être traitée rapidement. Par exemple, dans une connexion TCP utilisée pour un trafic entre un terminal interactif et un ordinateur, les caractères qui arrêtent et qui relancent la sortie standard (contrôle-s et contrôle-q) peuvent être considérés comme urgents puisque le récepteur doit les traiter immédiatement.

II.6.3.11. Les numéros de "port" réservés

TCP combine les liaisons d'accès statiques et dynamiques de la même façon que UDP, en utilisant un ensemble de numéros d'accès ou numéros de ports (courrier électronique etc.). Il laisse la majorité de ces numéros disponibles dans le système d'exploitation pour que les programmes d'application puissent les allouer en cas de besoin. Le tableau II.6.3.11. ci-dessous illustre les accès ou numéros de ports TCP affectés actuellement [Comer 88]:

Tableau II.6.3.11. Les numéros d'accès TCP:

Décimal	Mot-clé	Description
0		Réservé
1-4		Non assigné
5	RJE	Entrée à distance des travaux
7	ECHO	Echo
9	DISCARD	Abandonner
11	USERS	Utilisateurs actifs
13	DAYTIME	Journée
15	NETSTAT	Statistiques du réseau
17	QUOTE	Numéro de référence du jour
19	CHARGEN	Générateur de caractères
20	FTP-DATA	Protocole de transfert de fichiers (donnée)
21	FTP	Protocole de transfert de fichiers
23	TELNET	Connexion du terminal
25	SMTP	Protocole simple de transport du courrier
37	TIME	L'heure
39	RLP	Protocole de localisation des ressources
42	NAMESERVER	Serveur des noms des hôtes
43	NICNAME	Who ls
53	DOMAIN	Serveur du nom du domaine
67	BOOTPS	Serveur du protocole Bootstrap
68	BOOTPC	Client du protocole Bootstrap
69	TFTP	Transfert trivial de fichiers
75		Service privé d'appel téléphonique
77		Service privé RJE
79	FINGER	Finger
95	SUPDUP	Protocole SUPDUP
101	HOSTNAME	Serveur des noms des hôtes NIC
102	ISO-TSAP	ISO-TSAP
113	AUTH	Service d'authentification
117	UUCP-PATH	Le service des chemins UUCP
123	NTP	Protocole du temps réseau
133-159	Non assigné	
160-223	Réservé	
224-241	Non assigné	
247-255	Non assigné	

Malgré que les accès TCP et UDP sont indépendant, les concepteurs ont choisi d'utiliser les mêmes numéros de ports pour les services accessibles à la fois par UDP et TCP.

Chapitre II.7.

LE FONCTIONNEMENT DU COUPLE TCP/IP:

Communications entre applications

II.7.1. Objectifs et problèmes

II.7.1.1. Introduction

Les programmes d'applications sont construits au-dessus des protocoles de transport (TCP et UDP). Ce sont les applications de base de l'Internet, plus les applications propres aux système UNIX. Nous pouvons les découper en trois catégories:

1. les applications permettant de faire de la connexion à distance, telles que rlogin et telnet,

2. les applications permettant de faire des transferts de fichiers, telles que rcp (*remote copy)*, ftp (*file transfert protocol*) et tftp (*trivial file transfert protocol*), et

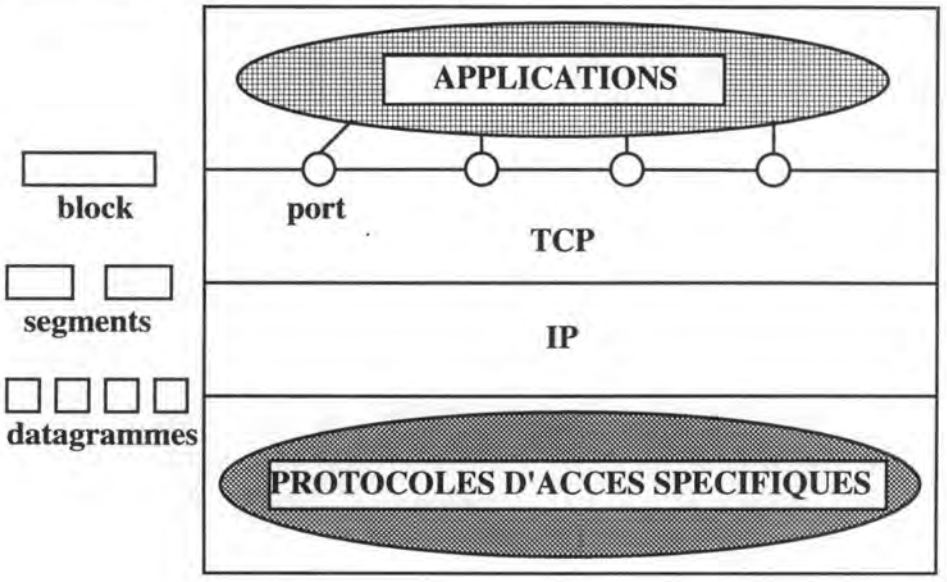
3. les applications permettant de réaliser des exécutions de programmes à distance, telles que rsh (*remote shell*).

Les programmes d'applications d'une machine communiquent avec les programmes d'application d'une autre machine via les différents couches du modèle TCP/IP. La couche TCP, par exemple, "discute" avec les applications en utilisant des blocs. Ces blocs passent entre les deux couches à travers des points spécifiques appelés "*sockets*".

Au niveau de la couche TCP, ces blocs sont adaptés à des formats de segments puis transmis vers la couche IP. La couche IP va également adapter les segments en un format de datagramme qui seront à leur tour transmis vers la couche inférieure de l'interface réseau.

La figure II.7.1.1.a. qui suit illustre les principes de TCP/IP:

Figure II.2.7.1.1.a. Principes de TCP:



Ainsi se passe une transmission dans le cas normale. Cependant, différentes problèmes peuvent surgir durant la transmission d'un datagramme:

- réseau saturé (noeud surchargé),
- réseau interrompu (coupure de ligne),
- erreur de transmission etc.

Au niveau transport (TCP) le service offert est matérialisé par une connexion transport entre deux sockets (= adresse IP + port) . L'adresse IP identifie une machine dans le monde entier et le port identifie une application sur cette machine. Le transfert de données entre les deux *sockets* est "*full duplex*".

II.7.2. Fonctionnement

II.7.2.1. Etablissement d'une connexion

Pour établir une connexion, on se base sur le protocole de transport TCP, développé au chapitre II.6. TCP étant un protocole orienté connexion, avant que le trafic à travers le réseau ne puisse avoir lieu, les deux applications qui veulent communiquer doivent se mettre d'accord pour ouvrir une connexion.

En fait, une ouverture de connexion est réalisée en deux temps: la machine qui demande l'ouverture de connexion passe un message appelé *Active Open* à l'entité IP, celle-ci livre ce message à l'entité TCP réceptrice. La machine réceptrice doit être en mode *Passive Open* pour que la connexion s'établie.

Par exemple, si une application sur une machine désire ouvrir une connexion avec une application sur une autre machine, soit l'application A. Cette dernière doit exécuter la requête

Passive Open , pour prévenir la couche inférieure qu'elle est prête à ouvrir une connexion avec une autre application qui en ferait la demande.

A partir de ce moment seulement, l'application émettrice, soit l'application B, peut exécuter la requête **Active Open** pour demander l'établissement de la connexion entre les deux applications.

La requête **Passive Open** peut prendre deux formes:

- soit l'application A n'est prête à accepter l'ouverture d'une connexion qu'avec une et une seule application, dans ce cas, elle effectuera la requête **Specified Passive Open**;
- soit elle est prête à accepter l'ouverture d'une connexion avec toute autre application, dans ce cas, elle effectuera la requête **Unspecified Passive Open.**

Pour établir la connexion, la couche TCP utilise la méthode "Three-way handshake" dont on a parlé dans le chapitre précédent, et ceci au cas où le réseau n'est pas fiable.
La connexion n'est ouverte qu'après trois acquittements:

1. Je demande à l'autre une connexion.

2. L'autre me répond qu'il accepte ma demande.

3. Je réponds à l'autre que j'ai compris qu'il l'a acceptée.

C'est seulement après ces trois acquittements que la connexion est effectivement ouverte entre les deux sockets et que la transmission de données peut commencer.

Les deux figures II.7.2.1.a. et II.7.2.1.b. ci-dessous illustrent l'établissement d'une connexion au cas où le service réseau est fiable et au cas où il ne l'est pas:

Dans la figure II.7.2.1.a. deux cas d'ouverture de connexion sont présentés; le premier cas est quand la machine réceptrice, soit B, est en attente et la machine émettrice, soit A, demande l'ouverture de la connexion. Dans ce cas, A envoie un *Active Open*, caractérisé par un "send" au niveau l'entité TCP émettrice et par un "deliver" au niveau de l'entité TCP réceptrice.

B étant en *Passive Open* elle envoie ce message synchronisé au niveau TCP qui fait un "send" via IP, et qui sera caractérisé par un "deliver" au niveau de l'entité TCP réceptrice A. Cet échange se fait en deux phase car le réseau est considéré fiable.

Le deuxième cas est lorsque les deux applications sur les deux machines essayent d'ouvrir une connexion entre eux en même temps; dans ce cas, deux *Active Open* sont générés des deux cotés et envoyés via IP en même temps. Un *Open Success* est reçu de chaque coté indiquant que l'ouverture est établie.

Dans chaque *Active Open* ou *Passive Open* un message est retourné vers l'application de l'entité TCP demandant l'ouverture de connexion, ce message est un *Open ID*. Il indique que le message d'ouverture (actif ou passif) est bien envoyé.

Le *Three-way handshake* sera utilisé dans la figure II.7.2.1.b. C'est le cas où le réseau n'est pas fiable. Nous pouvons voir les trois messages qui traversent les deux couches (TCP + IP) en l'occurence, le réseau.

La première application envoie un *Active Open*, l'autre application étant passive et la connexion est ouverte de son côté, l'entité TCP répond ainsi par un "ACKi + 1" indiquant l'accusé de réception de la demande d'ouveture de connexion.

Malgré que la connexion est ouverte, l'application qui a demandé l'ouverture envoie de nouveau un *Send* indiquant qu'elle est au courant de l'ouverture de la connexion et qu'elle l'a acceptée. A ce moment, la connexion est établie entre les deux applications.

Figure II.7.2.1.a.: Protocole d'établissement de connexion simplifié:
(Cas où le service réseau est fiable)

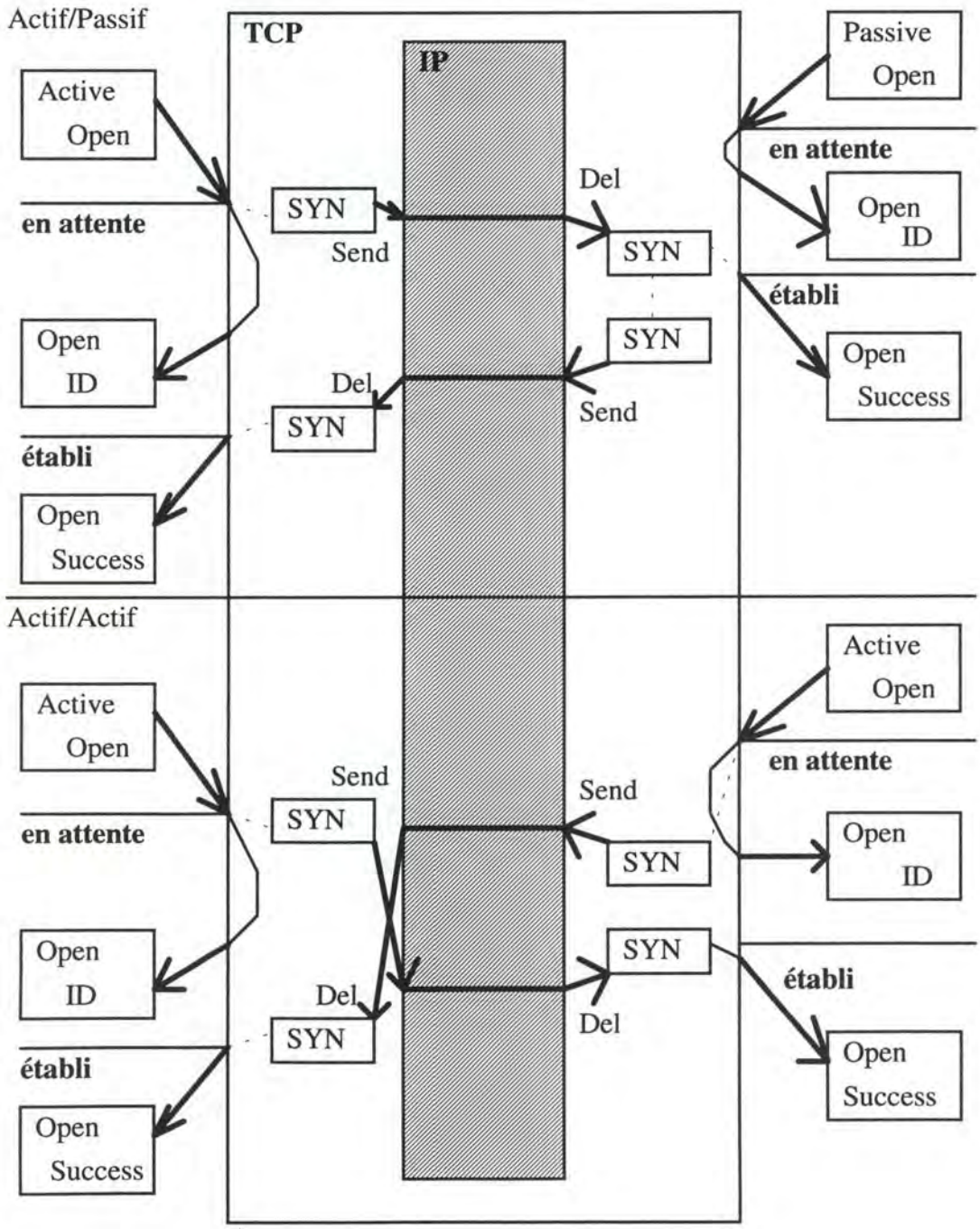
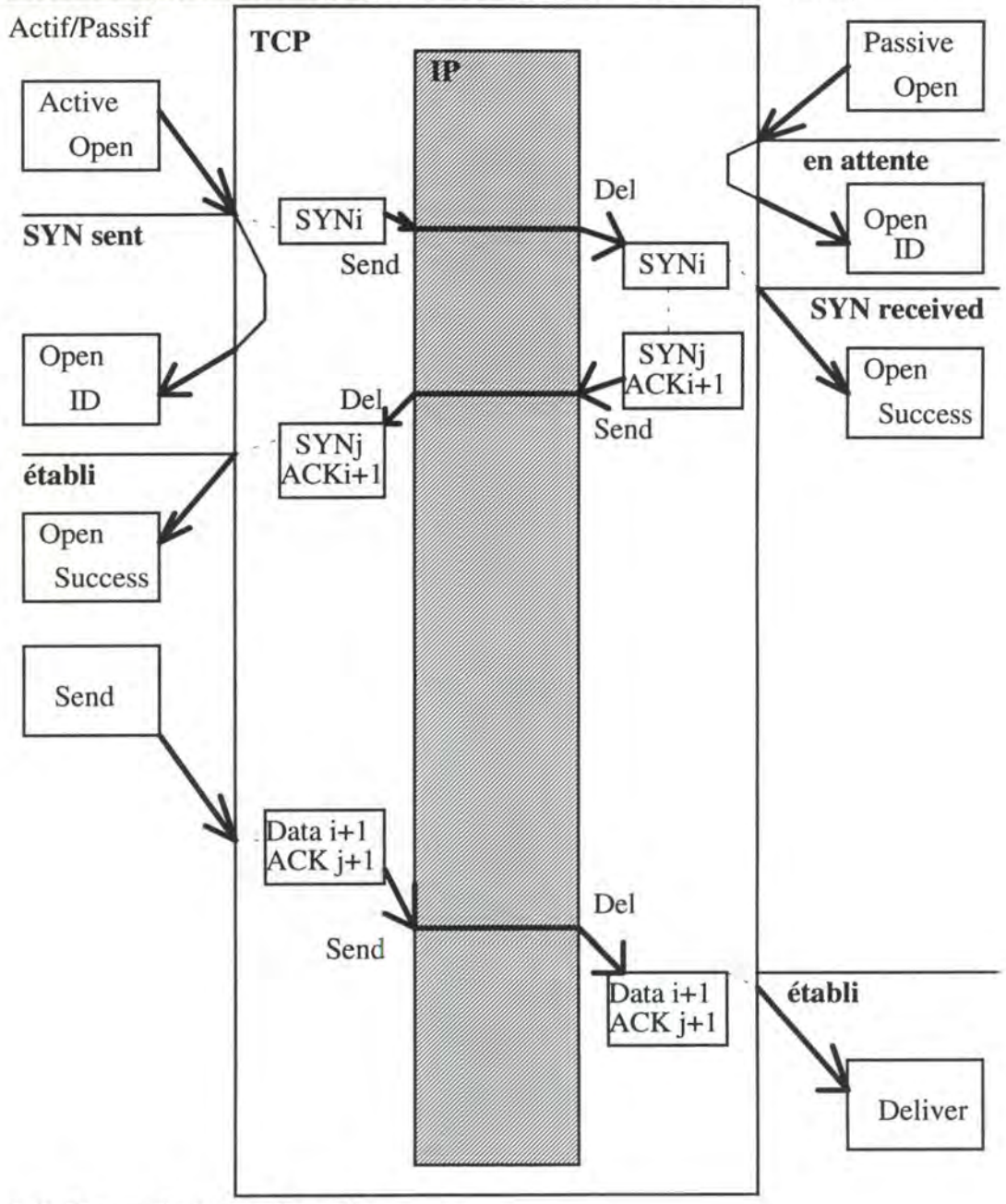


Figure II.7.2.1.b.: Protocole d'établissement de connexion: "Three-way handshaking" (Cas où le service réseau n'est pas fiable)

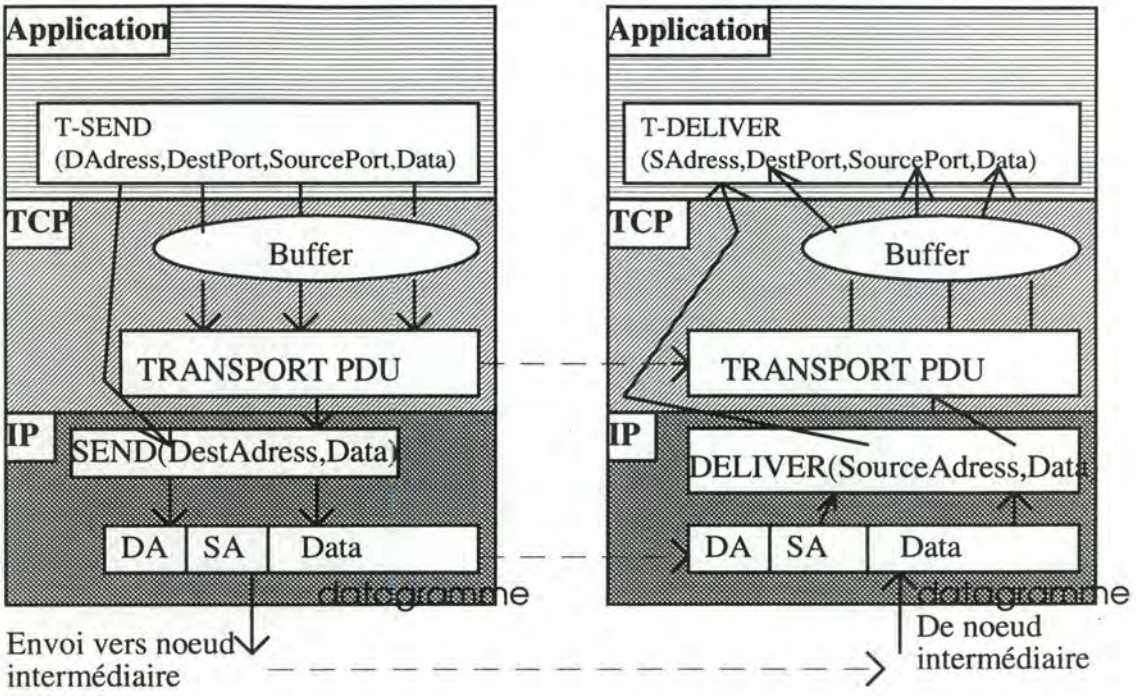


II.7.2.2. Transfert de données

Un segment peut se perdre, être corrompu, arriver hors-séquence ou bien arriver en retard, avec des délais variables et non prévisibles; pour faire face à ces problèmes des mécanismes très complexes sont mis en place:

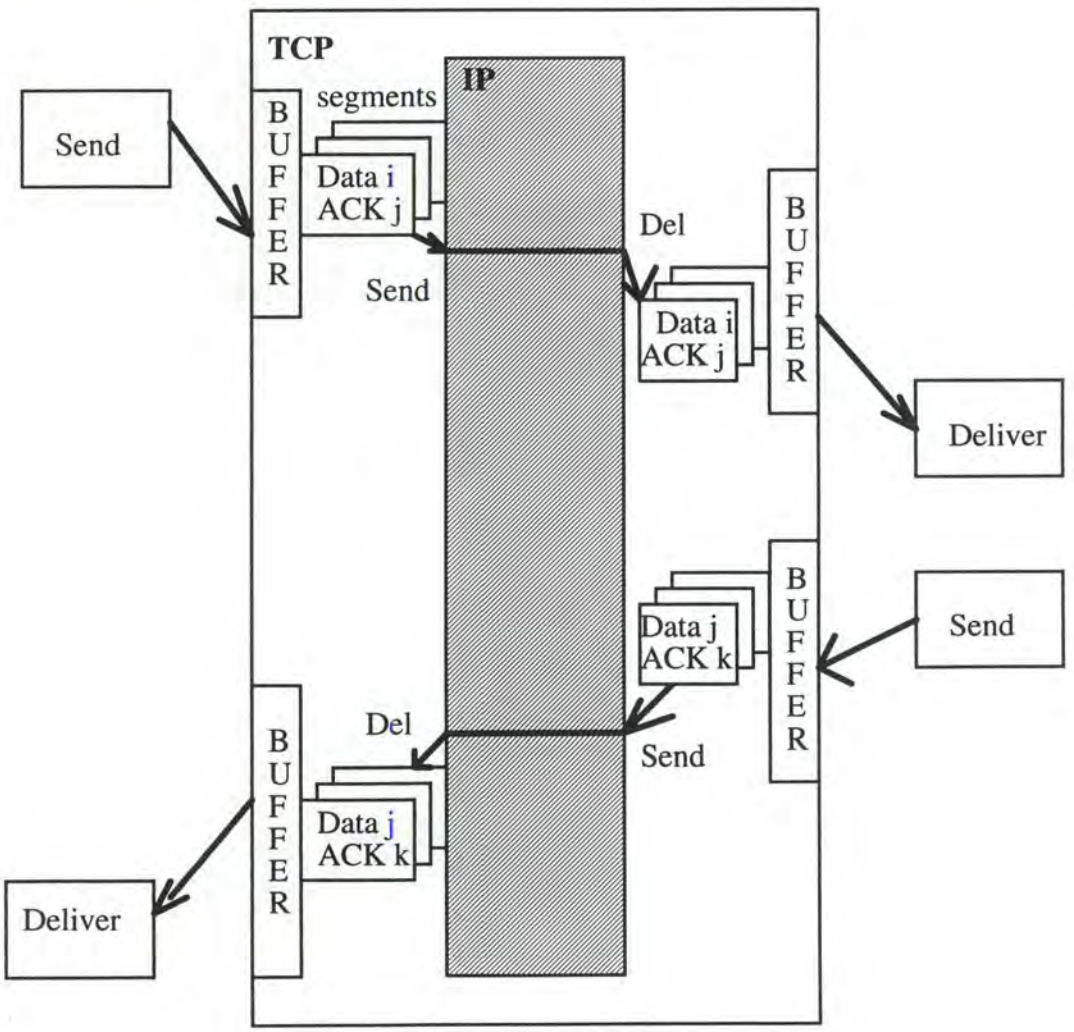
- Pour les pertes de séquences, les segments sont numérotés par l'émetteur et remis en ordre par le récepteur.
- Pour les pertes et corruptions, les segments sont acquittés par le récepteur. Ils sont retransmis par l'émetteur en cas d'attente trop longue de l'acquittement.

Pour chaque segment à transférer il y a un délai limite (*Timeout*). Si avant l'expiration du délai l'émetteur n'a pas reçu un acquittement de réception du destinataire, alors le segment est retransmis sauf si la transmission a déjà échouée un certain nombre de fois (dans ce cas la transmission est définitivement abandonnée). La figure ci-dessous illustre ce transfert:



La figure II.7.2.2. ci-dessus illustre le transfert de données:

Figure II.7.2.2.: Protocole de transfert de données:



Partie III: Implémentation

Chapitre III.1.

LE FICHIER NADF

III.1.1 L'adaptateur de format

III.1.1.1. Introduction

III.1.1.1.1. Objectifs

Le format des audit trails générés par la commande etherfind du système d'exploitation SunOS est non conforme à l'évaluateur ASAX. En effet, comme nous l'avons exposé dans le chapitre I.2, ASAX ne reconnaît que des fichiers de format standard nommé NADF. Par conséquent, l'implémentation d'un adaptateur de format pour les fichiers natifs (cf. Annexe D) se révèle indispensable. Un tel adaptateur a pour objectif la conversion de tout fichier de traces produit par le mécanisme d'audit (cf. Annexe C) en un fichier de format NADF.

III.1.1.1.2. Généralités sur les adaptateurs de formats

Un adaptateur de format est un module assurant la conversion d'un fichier structuré en un format approprié à l'outil d'analyse.

Dans le cadre de ce mémoire, le format d'origine est le format natif généré par la commande etherfind (cf. Annexe C) et le format cible est le NADF. D'autres fichiers auxiliaires doivent être également générés. Ces derniers garderont des information nécessaires à l'analyse (exemple: table de conversion de noms, etc.)

L'adaptateur de format s'exécutant sur un fichier natif ou de traces, génére le fichier NADF. Le fichier de description des données d'audit est généré à la main.

III.1.1.2. Structure du fichier NADF:Format des enregistrements

Un fichier NADF est un fichier binaire, séquentiel composé de records ayant un format standard appelé NADF (*Normalized Audit Data Format*). Ce dernier a été inspiré du format des audit records générés par BS2000 vu la flexibilité que celui-ci présente.

L'en-tête d'un fichier NADF est constitué d'une séquence de caractères: "__NADF__1\0", ce qui l'identifie parmi d'autres types de fichiers, le reste du fichier est une séquence de records.

Un record de format NADF possède la structure suivante:

1. real length: un champ de quatre bytes contenant la longueur (en bytes) réelle du record, à laquelle est ajoutée quatre bytes, longueur de la zone contenant le champ de la longueur;

2. le contenu du record: représente une séquence de champs appelés données d'audit (pour *audit data*). Chaque audit data est composé de:

   2.a. son identifiant: un entier représenté sur *deux bytes* et identifiant cet audit data parmi d'autres;

   2.b. sa longueur: un entier représenté sur *deux bytes indiquant la longueur de la valeur;*

   2.c. sa valeur: c'est la valeur de l'audit data, qui doit être alignée (nombre pair de bytes).

Exemple: L'identifiant du champ audit data "DA" est "111", ce champ est aligné sur deux bytes, le champ longueur contient la longueur de la donnée, et est aligné également sur deux bytes, enfin le champ valeur contient la donnée de ce champ, sa longueur doit être alignée sur un nombre paire de bytes. Si la longueur est impair, on ajoute un caractère "espace" pour aligner la chaîne de caractères de la donnée sur deux bytes et on augmente la longueur de 1.

Voici ci-dessous l'illustration du format NADF d'un record:

Format de l'enregistrement ou "*audit record*" de l'"*audit trail*".

$\forall$ i : 1 <= i <= n, id1 < id2 < ... < idi < ... <idn.

Où identifiant et longueur sont deux entiers de deux bytes, valeur aura un type qui sera fixé par l'identifiant et longueur totale est un entier "*long*" de quatre bytes. Le programme recevra un fichier de traces comme input et produira en output un fichier NADF qui est la transformation du format du premier fichier en un format normalisé.

Le format NADF est extrêmement simple et flexible. Le fichier NADF se compose d'une séquence d'enregistrements chronologiques (voir figure ci-dessus). Chaque enregistrement est composé de la longueur réelle de l'enregistrement et d'une liste d'audit data. Chaque donnée est représentée par un identifiant, la longueur de la donnée et la valeur de cette donnée dont le type dépend uniquement du déterminant. Les audits datas sont triés sur l'identifiant afin d'optimiser leur traitement:

III.1.1.3. Le fichier auxiliaire du fichier NADF: dsecription des audits data

Ce fichier correspond à un fichier séquentiel de type texte contenant des informations décrivant les audit datas du fichier converti.

La description de chaque audit data est constituée d'une séquence de cinq lignes: chacune commence par un numéro allant de 1 à 5. La sémantique de chaque ligne est décrite ci-dessous:

1. identifiant: l'identifiant d'un audit data constitue une contrainte essentielle: c'est un entier qui permet de repérer univoquement chaque audit data parmi tous les audits data existants. L'identifiant est représenté sur une ligne commençant par le chiffre "1";

2. type d'origine: il indique la représentation interne des audits data dans le fichier natif. Dans le but de réduire la taille des fichiers de traces, le mécanisme d'audit enregistre quelques audits data sous forme interne (appelée: forme codée). Le type d'origine est représenté sur une ligne commençant par le chiffre "2";

3. type de destination: il indique la représentation interne des audit datas dans fichier NADF. Actuellement, ASAX a un seul type, il interprète chaque audit data comme étant une chaîne de bytes (string of bytes). Le type de destination est représenté sur une ligne commençant par le chiffre "3";

4. nom externe: à la forme interne est associée une forme externe (une forme lisible), c'est le nom externe de l'audit data. Ce dernier est utilisé lors de la programmation d'une requête. Le nom externe est représenté sur une ligne commençant par le chiffre "4";

5. descripion sémantique: elle sert à illustrer brièvement la signification de l'audit data. La description sémantique est représentée sur une ligne commençant par le chiffre "5";

L'ensemble de description des audit datas est précédé par une séquence de six lignes spéciales constituant l'en-tête du fichier de description des audit datas. En voice la description:

1. la première ligne commence par la lettre "A" et est suivie du nom de la version du système d'exploitation des audit trails;

2. la deuxième ligne commence par la lettre "B" et est suivie par un ensemble d'informations déterminant les caractéristiques de la machine productrice des fichiers de traces;

3. la troisième ligne commence par la lettre "C" et est suivie par la date et l'heure auxquelles le fichier de traces à été créé;

4. la quatrième ligne commence par la lettre "D" et est suivi par le nom de la version du programme (adaptateur) générateur du fichier NADF;

5. la cinquième ligne commence par la lettre "E" et est suivie par des informations sur les caractéristiques de la machine productrice du fichier NADF;

6. la sixième ligne commence par la lettre "F" et est suivie par la date et l'heure pendant lesquelles le fichier NADF à été généré.

Voici ci-dessous la syntaxe BNF du fichier de description d'audits data:

<data description file>::=<header lines> '<data lines> =
<header lines>::=<empty lines>|<A_lines>|<B_lines>|<C_lines>|<D_lines>|<E_lines>|
<F_lines>

for a = A,..., F:
<a_lines> ::=<a_line> '<a_line>=
<a_line> ::=<empty_lines>|<a><blank><any_seq><eln>
<a> ::=a

<data lines> ::=<1_line><2_line><3_line><4_line>[<5_line>]

<1_line> ::=<empty_lines> 1 <blank> <integer_s> <spaces> <eln>

<integer_s> ::=0|1|2|...|65536

for x=2,3,4,:
<x_line> ::= <empty_lines> <x> <blank> <token> <spaces> <eln>
<x> ::= x
<token> ::= any sequence of letters, underscores and digits beginning with a letter or an underscore

<data_lines5> ::= <space> 5 <spaces> <free text>
<5_line> ::= <empty_lines> 5 <blank> <any_seq> <eln>

<empty lines> ::= any sequence of blanks tabs or end of lines
<blank> ::=
<spaces> ::= any sequence of blanks and tabs
<any_seq> ::= any sequence of characters excluding end of line

<eln> ::= the end of line character

Exemple:

1 111
2 short int
3 adresse
4 DA
5 destination_address

III.1.1.4. Description et implémentation de l'adaptateur de format

Dans le cadre de ce mémoire, l'analyse des fichiers de traces générés par la commande "etherfind" nécessite un passage par une première étape: la conversion du fichier de traces en un format NADF. Cette conversion est réalisée par un programme "adaptateur". La réalisation de ce programme à première vue paraît relativement facile: il s'agit de lire dans le fichier natif, de transformer les records natifs en format NADF et d'enregistrer les records convertis dans le fichier NADF, et ceci jusqu'à la fin du fichier natif.

L'adaptateur de format reçoit en entrée un fichier contenant des paquets hexadécimaux captés sur le réseau (cf. Annexe D) et génère en sortie le fichier converti NADF.

Cette section présente l'algorithme abstrait qui illustre le fonctionnement du programme adaptateur et la spécification des fonctions utilisées. Le code de l'adaptateur est disponible dans l'Annexe E.

Algorithme:

```
création_en_écriture_fichier_nadf;

ouverture_en_lecture_fichier_natif;

tant que pas_fin_fichier_natif faire

        {

                lire_paquet_natif;

                traiter_paquet_natif;

                ecrire_paquet_traité;

        }

fermer_fichier_natif;

fermer_fichier_nadf;
```

Fonctions:

La fonction principale *Main* reçoit trois arguments: argc = 3, argv[0] est le nom du fichier source, argv[1] est le nom du fichier de trace à analyser et à traduire, et argv[2] est le nom du fichier NADF à générer. Ces arguments représentent la précondition de la fonction. La postcondition est un fichier de nom argv[2] qui sera généré. Ce fichier correspond au fichier de nom argv[1] converti en format NADF.

La fonction *creat_NADF* reçoit comme argument le nom du fichier NADF à créer, ce qui constitut sa précondition. Si l'opération réussi (fd1 >= 0) la valeur retourné sera positive ou nulle et représente le descripteur de fichier NADF créé. Sinon la valeur sera négative. Cette valeur représente le code de l'erreur ayant eu lieu. Le résultat de la création constitut la postcondition de la fonction.

La fonction *trt_paquet* reçoit le pointeur fd1 du fichier à lire (fichier de trace), et les deux premiers nombres hexadécimaux du paquet à traiter. La lecture et le traitement du reste du paquet se fait dans cette fonction qui fait appel à d'autres fonctions. Sa précondition est réalisé par le positionnement sur le paquet à traiter et par le descripteur de fichier de trace. Sa postcondition est réalisé quand le paquet est traité et écrit dans le fichier NADF.

La fonction *trt_tcp* est appelée si le protocole utilisé est tcp. Elle reçoit le pointeur vers la dernière cellule traité (IP), la longueur correspondant au reste du paquet, elle doit retourner la longueur de la liste chaînée qu'elle vient de créer. Chaque cellule de la chaîne est composée de quatre champ: le premier est consacré à l'identifiant du champ du paquet. Cet identifiant est sous forme d'un entier représenté sur deux bytes. Le deuxième est consacré à la longueur de la valeur du champ du paquet. Cette longueur est également sous forme d'un entier représenté sur deux bytes. Le troisième est consacré à la valeur qui est, selon la pertinence du champ, convertie ou simplement copiée. Le quatrième est consacré au pointeur vers la cellule suivante ou nil.

La fonction *trt_udp* a la même spécification que *trt_tcp* sauf que les champs sont différents.

La fonction ecrire_buf reçoit la longueur totale et le pointeur vers la liste chainée. Elle retourne le pointeur ptr_rec qui va être passé par la suite à la fonction prédéfinie *write_NADF*. Sa précondition se réalise quand la liste chainée qui contient les champs du paquets est complétée. Sa postcondition se réalise quand elle passe le pointeur sur le buffer. Ce buffer contient les champs du paquet déjà sous format NADF.

La fonction *write_NADF* reçoit: le descripteur du fichier NADF ouvert avec *creat_NADF*, l'adresse du buffer contenant le paquet et qui va être écrit dans le fichier NADF, et un drapeau indiquant si le buffer doit être écrit sur le disque en bloc ou champ par champ. Ces arguments représentent la précondition de la fonction. La postcondition est réalisé avec l'opération d'écriture dans le fichier NADF. Si l'opération d'écriture se déroule bien, la valeur retournée est positive ou nulle sinon une valeur négative est retournée pour signaler le type d'erreur. Les codes d'erreurs se trouvent dans d'autres fichiers qui sont définis avec les fonctions prédéfinis de l'équipe ASAX dans les zones d'*include*.

La fonction *copy_bytes* reçoit un pointeur sur l'adresse source à partir de laquelle elle va copier des bytes, un pointeur sur l'adresse cible à partir de laquelle elle va commencer à écrire et le nombre de bytes n qu'elle doit copier (la zone d'écriture doit être alloué à priori d'une taille supérieure ou égale à n). Sa précondition est constituée des deux pointeurs et sa postcondition est réalisée quand n bytes sont copiés, dans l'ordre, à partir de l'adresse source vers l'adresse cible.

La fonction *close_NADF* reçoit le pointeur fd1 du fichier NADF. Elle a comme précondition ce descripteur qui indique l'emplacement du fichier ouvert par *creat_NADF*. Si l'opération réussi (fd1 >= 0) la valeur retourné sera positive ou nulle, sinon la valeur sera négative. Cette valeur représente le code de l'erreur qui pourrait se produire. Le résultat de la fermeture du fichier constitut la postcondition de la fonction.

D'autres fonctions comme lire_n_hexa, conv_hex, et conv_adr_ip ont des spécifications triviales. Le lecteur désirant plus de détails peut consulter les commentaires des fonctions dans le code (cf. Annexe E.).

III.1.2. Les problèmes rencontrés

III.1.2.1. Au niveau de la programmation

La commande etherfind (cf. Annexe C.) génére des fichiers ascii contenant les paquets captés sur le réseau selon les options choisies. Ces fichiers sont composés de paquets dont la structure varie selon le protocole utilisé (cf. Partie II). L'adaptateur de format doit d'abord se positionner sur le premier hexadécimal du premier paquet, pour cela il doit passer les lignes d'en-tête utilisées dans presque toutes les options et en particulier dans l'option -x (:format hexadécimal). Ces lignes présentent, selon l'option, le titre de la colonne imprimée. Une fois que le premier hexadécimal est atteind, le traitement commence. Pour un paquet TCP, par exemple, les six premiers hexadécimaux représentent l'adresse de destination de la machine (champ ajouté par le support ethernet) qui fait partie de l'en-tête de la trame (pas encore le paquet IP), ensuite l'adresse source également sur six hexadécimaux et le champ FT sur deux hexadécimaux. L'hexadécimal qui suit représente les deux premiers champs du paquet IP, (début de l'en-tête du paquet), généralement égale à 45 où 4 représente la version utilisée et 5 la longueur de l'en-tête, qui est pour ce cas égale à 20 bytes (5 mots de quatre bytes) car il n'y a pas d'option (cf. Partie II). Et ainsi de suite, nous parcourons tous les champs de l'IP puis, dans ce cas, tous les champs du TCP, à la fin il y a des hexadécimaux correspondant au champ CRC (*Cyclic Redundency Check*) qui n'appartient pas à IP mais plutôt à la trame du réseau.

Pour distinguer, par exemple, un paquet TCP d'un paquet UDP, il suffit d'analyser le champ "proto" qui est égale à 06 pour les TCP et 11 pour les UDP. les paquets traités sont de deux types: TCP et UDP. Généralement, ce sont les seuls paquets qui renferment des données utilisateur, donc ces sont les plus pertinents pour l'analyse.

Le champ "proto" se trouve dans l'en-tête IP. Pour l'analyse il faut traiter le champ longueur totale et le champ proto avant de pouvoir appeler les fonctions trt_udp ou trt_tcp. Les champs sont ainsi traités byte par byte, une liste chaînée est créée à fur et à mesure et est remplie par les informations correspondantes jusqu'à l'arrivée aux données.

Les spécifications de cette liste chaînée sont utilisés dès le début car il n'y avait pas de spécifications précise sur l'utilisation des fonctions prédéfinies établies par l'équipe ASAX, ni sur les paramètres à échanger. Pour cela les données sont traitées et stockées dans cette liste.

Le passage de la liste au buffer a été résolue par l'implémentation d'une fonction supplémentaire "ecrire_buf" qui fonctionne correctement.

En fait, il ya un compteur qui calcule la longueur de la liste. Chaque liste correspond au traitement d'un paquet; après le traitement, la liste liste est copiée dans un buffer qui sera écrit dans le fichier binaire NADF en respectant le format. A chaque écriture, le buffer contient un paquet et sa longueur totale. le format de ces paquets est présenté dans le chapitre III.1.

Le champ donnée est le champ le plus pertinent dans un paquets. Les seuls paquets contenant des données sont les paquets TCP et les paquets UDP. Un filtrage est déjà exécuté à ce niveau dans l'adaptateur. Seuls les paquets TCP et UDP seront traités et donc disponibles dans le fichier NADF pour des analyses ultérieures.

En fait, la difficulté qui se posait était de passer un pointeur sur le buffer contenant un paquet à une fonction prédéfinie qui est *write_NADF*. Cette fonction développé par l'équipe ASAX n'est pas la seul disponible pour l'adaptateur. D'autres fonctions telles que *creat_NADF*, *open_NADF*, *read_NADF* et *close_NADF* sont également utilisable pour différents type d'adaptateurs. La complexité des structures des fichiers de traces contenant les paquets ne nous a pas permis d'utiliser la fonction *read_NADF*. L'analyse champ par champ et de façon plus approfondie byte par byte nous a obligé à implémenter des fonctions spécifiques que le lecteur peut consulter dans l'Annexe E.

III.1.2.2. Au niveau de l'analyse *on-line*

Une analyse *on-line* est possible: plutôt que de générer le fichier NADF et de l'évaluer par après (analyse *off-line*), chaque paquet généré est converti et évalué à fur et à mesure. Ce processus continue nécessite un évaluateur qui analyse des paquets (en continu) et non des fichiers, ce qui n'est pas le cas pour l'instant. La réalisation d'un tel outil serait possible si on procéde à des modification plus ou moins importante de l'outil ASAX.

En effet, pour l'analyse *on-line* l'adaptateur de format doit lire dans l'input standard plutôt que dans un fichier. Dans ASAX, il a été prévu un moyen pour rendre le processus d'évaluation tout à fait indépendant du mécanisme de saisie des enregistrements d'audit. L'utilisateur devrait donc implémenter ses propres routines d'entrée/sortie pour gérer lui-même la façon dont les enregistrements d'audit sont soumis à l'évaluateur. Pour l'analyse *on-line,* la lecture se fait à partir de l'input standard; dans l'analyse du mode inversion/évaluation, où le fichier de traces est directement évalué, un enregistrement est lu dans le fichier de traces, converti en format NADF et puis transmis à l'évaluateur (pas de fichier NADF intermédiaire).

Pour cacher ces divers modes de saisie des audit records, dans ASAX, on a introduit la notion de fichier "virtuel" ainsi que des opérations de manipulation de tels fichiers: *v*open, *v*read, *v*close. Le mot "virtuel" exprime le fait que les opérations précitées agissent sur un fichier d'audit virtuel. Ce dernier peut être un simple fichier de traces, mais en général, il correspond

à un flot séquentiel d'enregistrements d'audit résultant d'un ensemble d'opérations arbitraires (collection, sélection, conversion, fusion,...) sur un ensemble de fichiers de traces.

En résumé, l'utilisateur doit fournir un seul module objet implémentant trois routines: *v*open, *v*read, *v*close. Ce module est ensuite intégré avec le noyau d'ASAX. En réalité, ce module est soumis à un outil automatisé (**masax**) faisant partie de l'environnement ASAX. Cet outil permet de générer une version exécutable d'ASAX correspondant à ces routines.

Pour accomplir cette analyse, le programme source de l'adaptateur va se réduire en un module ayant trois routines appelées: *v*open (*virtual open*), *v*read (*virtual read*) et *v*close (*virtual close*), pour être intégrées avec le noyau d'ASAX. Ceci ne fait pas partie de ce mémoire, mais il peut être intéressant de l'implémenter dans un projet futur.

Chapitre III.2.

LES REQUETES RUSSEL

III.2.1 L'analyse des fichiers de traces

III.2.1.1. Introduction

Les règles RUSSEL sont utilisées pour faire des filtrages au niveau des champs de données. Ces filtrages peuvent être combinés pour donner des informations précises sur un événement. La reconstitution des informations récoltées peuvent nous aider à détecter des attaques.

Au niveau de la commande Etherfind, les paquets qui sont captés peuvent contenir des informations internes au réseau. Les paquets TCP contiennent généralement un seul caractère. Ces caractères peuvent être concaténés pour ainsi reconstituer un message. Ce message peut être un *login name*, si l'utilisateur veut se "logger" sur une machine, il peut être également un mot de passe ou toutes autres informations pertinentes.

Le filtrage peut se faire selon le type de surveillance souhaité. Si on désire contrôler le port FTP, par exemple, pendant une certaine période, il suffit de sélectionner parmi tous les paquets captés, pendant cet intervalle de temps, le port numéro 21 comme port de destination, ainsi nous pouvons visualiser toutes les informations sur ceux qui ont essayé de se connecter sur ce port.

Nous allons essayer d'implémenter d'abord une requête qui va afficher tous les champs des paquets captés. Cette requête s'appellera "display". Elle fera appelle à une routine externe écrite en langage C "*display_current*", l'interface de cette routine avec le langage RUSSEL va être expliqué dans le paragraphe qui suit.

Nous avons également voulu présenter quelques requêtes assez simples mais parfois pertinentes pour montrer toutes les possibilités de filtrage au niveau byte. Ces filtrages, comme nous l'avons déjà indiqué, sont de très bas niveau. Nous manipulons des hexadécimaux qui doivent être par la suite convertis par l'intermédiaire d'autres routines C.

Nous pouvons imaginer différentes sortes d'utilisations de cet outil, par exemple, pendant l'analyse des paquets générés par la commande etherfind, l'utilisateur peut garder dans des tables les adresses utilisées par certaines machines et surtout les numéros de ports. Ces informations peuvent constituer un profil de cette machine. Certaines machines sont utilisées par plusieurs utilisateurs mais ceux-ci peuvent être du même groupe.

Par exemple, nous pouvons surveiller une machine du pool des étudiants et en particulier le port FTP. Les paquets qui seront générés peuvent reconstruire tout le scénario de connexion y compris les mots de passe et les *login_names* des étudiants qui ont importés des fichiers sur ce port (pendant toute la nuit). Ceci n'est qu'un simple exemple d'utilisation et nous pensons qu'il y a plusieurs possibilités de filtrage.

Nous pouvons également générer des rapports statistiques sur l'utilisation du système. Ces rapports pourront contenir, par exemple, la liste de toutes les machines qui ont utilisés le réseau pendant un certain interval de temps, la liste des ports utilisés par chacune d'elles, ainsi que le nombre de connexions effectuées sur un port particulier par toutes les machines. Ces règles feront éventuellement appel à des routines C prédéfinies.

III.2.1.2. L'interface avec le langage C

Des routines en langage C peuvent être appelées par les règles. Pour cela, l'interface est assez simple. Cependant plusieurs tâches doivent être exécutées par l'utilisateur pour reconstruire le système qui va inclure ces nouvelles routines. Plusieurs outils sont fournis à l'utilisateur pour automatiser ces tâches. La nouvelle version de l'exécutable du système va alors reconnaître ces nouvelles routines et va donc les exécuter correctement.

Une fois que les routines C sont écrites, leurs noms et leurs paramètres et le type retourné doivent être introduits dans un fichier texte particulier de description. Ce fichier contient ainsi toute une librairie de fonctions prédéfinies. Chaque entrée de ce fichier est un descripteur d'une routine de la librairie.

Au lieu d'aller modifier les fichiers sources d'ASAX, quand on veut ajouter des nouvelles routines C à la librairie existante, un langage simple est fourni à l'utilisateur lui permettant d'introduire ces modifications et ce sont les outils d'intégration d'ASAX qui vont reconstruire le nouveau fichier exécutable ASAX.

Le passage d'arguments entre les routines C et l'évaluateur ASAX doit respecter des spécifications très précises. Les routines C qui doivent être intégrées dans la librairie, doivent fournir les arguments selon le format défini. Ceci est nécessaire pour que l'évaluateur puisse bien comprendre les appels. Les zones des arguments fournis par l'évaluateur doivent être correctement interprétés par les routines C pour éviter tout genre de disfonctionnement [Asax 94].

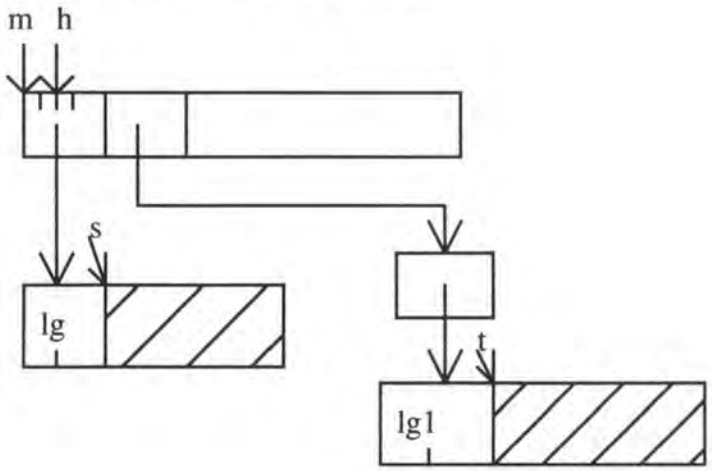
Généralement, les routines C utilisent plusieurs arguments. Dans ce cas, au lieu de passer une liste de tous ces arguments, un pointeur vers une zone mémoire contenant cette liste sera utilisé. Dans le cas, par exemple de la routine "*convert*" qui va être utiliser par la requête "*display*" (cf. Annexe F.) pour l'affichage des paquets, la sous-routine "*sub_conv*" reçoit un

pointeur s sur la zone de caractères à convertir ainsi qu'un entier lg qui indique la longueur de cette zone. Elle retourne un pointeur t sur la zone de caractère qu'elle a convertit ainsi que la longueur lg1 de cette zone qui est un entier.

Les deux arguments passés sont de deux types: string et entier. Pour l'entier le passage se fait par valeur. Cependant, pour le string le passage se fait par référence.

La routine convertir(m) (cf. Annexe F) avec l'argument m qui est le pointeur vers la zone mémoire allouée est appelée dans "*display*". Le pointeur passé en argument nous indique les adresses des arguments en entrée et en sortie. La structure de cette zone est très particulière. Les deux premiers bytes de la zone mémoire contiennent un pointeur vers l'adresse des arguments en entrée (s et lg). Les deux bytes qui suivent contient un deuxième pointeur qui pointe vers une autre adresse dans laquelle il y a un deuxième pointeur (pointeur de pointeur) vers la zone où on doit écrire les données converties ainsi que la longueur de la zone. La figure ci-dessous illustre cette combinaison de pointeurs.

La zone mémoire des arguments:



Une fois que les pointeurs sont initialisés, la sous-routine peut être appelée, elle doit garnir la partie lg1 ainsi que la zone pointée par t en lisant les données brutes à partir de la zone pointée par t et jusqu'à une longueur lg. Cette routine utilise également une table de correspondance "nadf.tab.h" qui fournit les noms des champs ainsi que leurs types (cf. Annexe F.)

III.2.1.3. Les règles RUSSEL

La première requête que nous présentons dans ce mémoire est celle qui affiche le contenu des champs des paquets captés (cf. annexe F.). Cette requête pourrait être modifiée pour être utilisée de différentes manières selon qu'on veut filtrer des champs ou en afficher la totalité. Elle utilise une variable globale v qui jouera le rôle d'un compteur de paquet. La routine C "*display_current*" (cf. annexe F.) affiche les champs du paquet sous un format spécifique indiqué dans l'annexe A. En effet, le fichier NADF étant un fichier binaire, les informations doivent être converties en des caractères lisibles pour l'analyse. Cette requête étant la première étape du traitement du paquet.

Les autres requêtes sont semblables bien que plusieurs filtrages sont effectués pour rendre l'analyse plus précise. C'est le cas de la deuxième requête qui sélectionne des données en filtrant sur le protocole TCP (06) et sur les deux adresses IP source et destination. Nous pourrons implémenter une petite routine C qui recevra en entrée ces adresses qu'on désire surveiller.

La règle *SELECT_DONNEE* testera ainsi les champs de chaque paquet et affichera la donnée en format hexadécimale puis convertie en caractères lisibles ainsi que les numéros de ports source et destination sélectionnés et les adresses IP choisies. Cette règles étant très simple, mais cette simplicité peut être rendue complexe si l'analyste veut pousser plus loin dans les filtrage.

La règle *"reconstruct"* similaire à *SELECT_DONNEE*, sauf que les données captés peuvent provenir de paquets du type TCP ou UDP. Le nom Data avec D majuscule affiche les données UDP (qui sont généralement illisibles). Cependant, le nom data affiche les données du paquet TCP qui est constitué, généralement, d'un seul caractère lisible.

Le résultat de ces règles est parfois assez compliqué à analyser. En effet, il faut reconstituer les messages récoltés pour pouvoir détecter les anomalies d'utilisation du réseau et du systèmes ainsi que le fonctionnement interne du système.

En fait, plusieurs sortes d'informations passent sur le réseau. Il y a des informations qui sont pertinentes et d'autres qui ne le sont pas. Les paquets qui fournissent des informations intéressantes et surtout les données sont les paquets de transport TCP et UDP. Les autres paquets tels que ARP, RARP, ICMP etc. peuvent parfois être intéressants, mais en général ils sont nécessaire seulement pour le fonctionnement interne du système.

Conclusion

Pour être sécurisés, les systèmes informatiques doivent être dotés d'un système d'alarme et d'un verouillage efficace, rendant ainsi la tâche difficile à toute utilisation malicieuse d'un utilisateur non autorisé. En effet, depuis longtemps les systèmes informatiques ont fonctionné sans faire appel à un système particulier de sécurité. Cependant, les ordinateurs étaient, au mieux, pourvus d'un mot de passe.

De nos jours, les ordinateurs manipulent des données parfois très sensibles qui sont véhiculées la plupart du temps à travers des réseaux informatiques. La question de la sécurité des systèmes informatiques devient alors un sujet principal dans la plupart des organisations informatiques. A cet égard, plusieurs d'entre elles se sont penchées sur le problème de la sécurité, à leur tête, le département de la Défense des Etats-Unis. Ce dernier a défini des règles de sécurité allant du niveau de sécurité minimal jusqu'au niveau auquel la sécurité du système doit être prouvée formellement.

Les machines Sun connectées sur le réseau Ethernet de l'Institut satisfont au niveau de sécurité C2. Le projet ASAX a pour objectif de définir et d'implémenter un système expert capable de faire une analyse universelle, efficace et puissante de tout fichier de traces pour ainsi atteindre le niveau de sécurité B3.

Le système d'exploitation SunOS est doté d'un mécanisme d'audit qui consiste à enregistrer dans des fichiers de traces les informations qui circulent sur le réseau Ethernet. Ces fichiers doivent faire l'objet d'un analyse adéquate pour déceler les scénarios d'attaque. Cette analyse nécessite un outil automatisé, efficace et "intelligent". ASAX en est un, c'est un outil qui permet d'évaluer tout fichier séquentiel structuré de format normalisé NADF, il dispose d'un langage efficace de règles "RUSSEL" permettant l'expression de requêtes d'analyse. Un fichier de trace généré par la commande etherfind est converti en un format NADF. La conversion est assurée par un adaptateur de format réalisé au cours de ce mémoire. L'analyse et le contrôle des différents événements tels que les scénarios d'attaques est réalisable par l'implémentation de règles en RUSSEL. L'application d'ASAX pour la détection des anomalies ou des utilisations suspectes du système a été vérifiée par l'élaboration de règles permettant de telles détections. Les paquets captés peuvent nous donner des informations supplémentaires sur les utilisateurs tels que les ports qu'ils ont utilisé, les comportements des machines et des ports (Ex. le port FTP), les machines sur lesquelles ils essayent de se connecter ainsi que leurs mots de passe, etc. Des statistiques concernant l'utilisation du réseau sont également possibles. La faisabilité d'un tel outil de contrôle appronfondi est possible.

Il serait intéressant, dans les projets futurs, de pouvoir faire des analyses *on-line* des fichiers de traces captés sur le réseau. Ainsi, on pourrait détecter des actions paraissant, a priori, normales, mais qui peuvent révéler un scérario d'attaques.

L'analyse des fichiers de traces sont qu'une solution parmi plusieurs qui contribue à la sécurité des systèmes informatiques. Elle n'est pas la meilleure solution car, en général, l'utilisateur malveillant ou l'intrus essaie toujours de trouver de nouveaux moyens pour contourner les mécanismes de sécurité mis en place. Ainsi, il faut disposer d'un mécanisme ayant une grande flexibilité pour contrer toutes sortes d'attaques nouvelles. Ce qui est, pour l'instant, difficile à concevoir.

Glossaire

Accusé de Réception ou Acquittement (*Acknowledgment*)
L'accusé de réception est l'ensemble de codes renvoyés par un système récepteur pour indiquer à l'émetteur que le message transmis a été bien reçu; il correspond à une étape dans le processus de transmission: le message a été totalement reçu, mais en aucun cas l'accusé de réception ne concerne le contenu du message. Dans la plupart des applications de communication de micro-ordinateur, l'acquittement est inséré dans la trame des signaux que l'émetteur attend pour continuer sa transmission. La gestion des acquittements est assurée respectivement par les deux systèmes en cours de transmission. Dans le cas de réseaux de transmission tel que Transpac, ces procédures sont prises en charge et déclenchées par le réseau lui-même.

Adaptateur (*Adapter*)
L'adaptateur est un dispositif permettant le raccordement d'un terminal à un réseau. Cet équipement, qu'il soit fourni par l'opérateur ou par l'installateur du réseau de communication, sert à assurer la mise en conformité des informations émises par le système émetteur aux caractéristiques de la ligne de ce réseau et réciproquement.

ANSI
American National Standards Institute. Organisation de normalisation américaine. C'est le seul membre de l'ISO (*International Standards Organization*) à représenter les Etats-Unis.

Baud
Un baud est une unité mesurant la rapidité de modulation. Une ligne téléphonique peut supporter 2400 changements d'état d'un signal par seconde, soit 2400 bauds. Pour offrir des débits de données plus forts, chaque changement peut représenter plusieurs bits: 2.4 ou plus au lieu d'un seul. Le nombre de bits par seconde s'obtient en multipliant le nombre de bauds par le nombre de bits de chacun de ces codes.

Bus
Ensemble non bouclé de canaux de transmission rapide de données reliant deux équipement informatiques, un bus est aussi l'ensemble des canaux de transmission reliant les différents modules du micro-ordinateur. Un bus est caractérisé par son débit et par le nombre de canaux parallèles reliant ces modules. Dans le monde des réseaux locaux, il existe plusieurs architectures de bus: bus Ethernet, bus à jeton (*token*), etc.

CCITT
Le CCITT (comité Consultatif International Télégraphique et Téléphonique) est un organisme international dépendant de l'UIT (Union Internationale des Télécommunication), sous l'égide de l'ONU. Son siège est à Genève. Ses membres sont les exploitants de réseaux publics, qu'ils soient administrations ou entreprises. Tous les quatre ans, l'assemblée plénière adopte des recommandations élaborées par des commissions chargées chacune d'un domaine déterminé.

Datagramme (*Datagram*)
Un datagramme est un bloc ou paquet d'information, transmis dans un réseau, sans référence ou sans ordre par rapport aux blocs précédents. Un datagramme contient en plus des données d'adresse du destinataire un numéro d'ordre. Pour être utile, il nécessite un dispositif de réassemblage dans l'équipement de destination du message.

Discretionary access Control
Un moyen de restriction d'accès à des objets, basés sur l'identité des sujets et/ou groupes auxquels ils appartiennent. Les contrôles sont discrétionnaires dans le sens ou un sujet avec une certaine permission d'accès est capable de passer cette permission à un autre sujet, directement ou indirectement.

Drapeau (*Flag*)
En télécommunication, c'est l'appellation particulière de l'ensemble des bits servant à délimiter un bloc de données. On l'appelle également "fanion" ou "délimiteur" ou encore "*flag*".

Emetteur-récepteur ou transmetteur (*transceiver*)
Equipement qui transmet, sans le modifier, un signal du micro-ordinateur (ou terminal) au câble et, réciproquement, dans un réseau local. C'est généralement un composant qui est situé à l'intersection du câble desservant le micro-ordinateur et du câble principal du réseau local, comme, par exemple, le câble coaxial d'un réseau Ethernet. Il existe des transmetteurs mono et multimédias.

Evénement (*event*)
Opération effectué par un sujet sur un objet et ayant un certains résultat.

FDDI (*Fiber Distribution Data Interface*)
Ce standard, établi par l'ANSI, est basé sur des supports en fibre optique, il a un taux de transfert de données de 100 Mbps, la longueur du réseau FDDI est limitée à environ 200 km, et le mécanisme de contrôle d'accès utilise la technologie du "*token ring*".

Fibre Optique (*Optical Fibre*)
Fibre constituée d'un "cheveu" fabriqué à partir de la silice ou de matière plastique, utilisée comme guide, à faible affaiblissement d'une onde de lumière modulée par des signaux de communication. Les câbles à fibre optique offrent des performances supérieures à celles des câbles coaxiaux, une meilleure qualité de transmission due à l'insensibilité aux parasites électriques et magnétiques, et un affaiblissement moindre. La distance entre deux répéteurs est de l'ordre de 40 km, contre environ 2 km pour le coaxial.

FTP (*File Transfer Protocol*)
C'est un protocole de haut niveau qui permet le transfert de fichiers d'une machine à une autre. Habituellement, il est implémenté comme étant un programme de niveau application, FTP utilise les protocoles Telnet et TCP. L'utilisateur doit fournir au serveur un identificateur (*login name*) et un mot de passe pour que celui-ci réponde à sa requête.

Handshaking
Expression anglo-saxonne pour décrire les procédures de mise en accord de deux équipements connectés.

IEEE (*Institute of Electrical and Electronic Engineers*)
Cette association d'ingénieurs en électronique américains joue un rôle important dans la normalisation à travers ses forums et ses travaux. Elle a notamment joué un rôle prépondérant dans la normalisation des réseaux locaux avec les normes IEEE 802 (802.3: Ethernet, 802.4: bus à jeton, 802.5: anneau à jeton).

ISO (*International Standards Organization*)
L'ISO est l'organisme international dépendant de l'ONU chargé de la normalisation. Il regroupe les instituts nationaux de normalisation de 89 pays qui sont répartis en 72 comités membres (par exemple AFNOR pour la France, BSI pour la Grande-Bretagne, DIN pour l'Allemagne) et 17 membres correspondants. Les résultats des travaux techniques de l'ISO sont publiés sous la forme de normes internationales.

Intrusion
Concept de sécurité décrivant la possibilité pour un utilisateur non habilité à se connecter à un système ou à un réseau de communication. On développe des techniques "anti-intrusion" pour s'en protéger. Elles utilisent des mots de passe, et des codes d'identification qui, selon les degrés de protection recherchés, sont plus ou moins complexes.

LAN (*Local Area Network*)
C'est un réseau physique qui opère à des vitesses très élevées (de dizaines de Mbps jusqu'à plusieurs Gbps) sur des distances courtes (jusqu'à quelques milliers de mètres). Ethernet est le support du LAN utilisé à l'Institut.

Logiciel (*Software*)
Ensemble de programmes, comme les compilateurs, les utilitaires, les applications du type d'un traitement de texte ou d'un système de gestion de bases de données, exécutables sur un ordinateur. Le logiciel est souvent opposé au matériel (*hardware*).

Logiciel de Communication (*Communications Software*)
C'est le logiciel qui, en plus du modem ou de l'interface de connexion à un réseau, va servir à établir une communication et gérer le dialogue avec le serveur ou d'autres postes de travail. Ce logiciel donne, en particulier dans le cas des télécommunications, des commandes au modem et des instructions de numérotation. Normalement, un logiciel de communication donné n'est pas assujetti à l'emploi d'un modem particulier ou à un serveur. Dans certains cas, pour optimiser ou mieux adapter sa communication avec un serveur donné, il peut être intéressant d'utiliser le logiciel proposé par le serveur.

Mainframe
Terme anglais désignant un site central ou gros ordinateur.

MAN (*Metropolitan Area Network*)
Réseau étendu au niveau d'une ville, obtenu par l'interconnexion de plusieurs réseaux locaux d'entreprises (RLE). Les MAN's émergent comme des services de transmission et de commutation pour un ensemble de bâtiments dispersés dans un rayon de 50 km dans une ville.

Un MAN englobe plusieurs RLE (LAN, *Local Area Network*). Plusieurs MAN peuvent former un WAN (*Wide Area Network*). Une norme est en cours d'étude (Comité IEEE 802.6), à partir des propositions de *Burroughs* et *National Semiconductor* d'un côté, et de *Telecom Australia* de l'autre.

Mandatory Access Control:
Un moyen de restriction d'accès à un objet basé sur la sensibilité de l'information contenue dans les objets et l'autorisation des sujets d'accéder à de telles informations.

Modèle ISO (*OSI Model*)
Appelé souvent modèle OSI, du sigle correspondant en anglais (*Open System Interconnection*), le modèle ISO (Interconnexion des systèmes Ouverts) est un modèle de référence, pour la communication de données entre deux systèmes, développé par l'ISO (*International Standard Organization*), en relation avec le CCITT (Comité Consultatif International pour le Télégraphe et le Téléphone). Ce modèle présente sept couches (appelées quelquefois niveaux) qui traitent chacune une étape du processus de communication entre deux systèmes.

Niveau de sécurité (*Security level*)
La combinaison d'une classification hiérarchique et d'un ensemble de catégories non hiérarchiques qui représentent la sensibilité des information.

Objet
Une entité passive qui contient ou qui reçoit de l'information.

Paire torsadée (*Twisted Pair Wiring*)
Deux fils de cuivre gainés et torsadés l'un avec l'autre (il s'agit d'éviter d'éventuelles perturbations électromagnétiques) forment un canal de transmission. Employée pour le câblage du téléphone dans les entreprises, la paire torsadée est devenu un support très utilisé, y compris dans les réseaux locaux informatiques. Un avantage essentiel de la paire torsadée à l'intérieur de l'établissement est de réduire la facture du câblage et de permettre l'utilisation des câbles déjà posés.

Paquet (*packet*)
C'est l'unité d'un ensemble d'informations de taille fixe transmises comme une entité de base dans un réseau à commutation de paquets. En plus des données, sous forme d'octets, le paquet se compose d'informations de service, nécessaires à son identification et à son acheminement. Une fonction PAD (*Packet Assembler Disassembler*) est nécessaire pour former les paquets et reconstituer symétriquement l'ensemble des données dans leur forme primitive. Cette fonction est assurée par un logiciel, module de communication de l'ETTD (Equipement Terminal de Traitement de Données) qui peut être fourni par le réseau Transpac ou directement implanté sur l'ordinateur au moyen d'une carte ou d'un coffret.

Passerelle (*Gateway*)
Equipement (logiciel et/ou matériel) permettant de transformer l'ensemble des procédures et protocoles d'un réseau dans celles d'un autre réseau différent pour leur permettre de communiquer. A la différence d'un pont qui se borne à faire passer les données d'un réseau vers un autre sans les adapter, la passerelle vise à adapter la syntaxe des informations d'un

système vers l'autre. Dans le modèle ISO (Interconnexion de Système Ouvert), une passerelle intervient jusqu'au niveau 6 (présentation), alors que le pont agit au niveau 2 (liaison de données), et le routeur au niveau 3 (réseau). Les passerelles sont souvent organisées en serveurs de communication.

Pont (*Bridge*)
Dispositif servant à relier entre eux plusieurs réseaux locaux, soit pour en agrandir les dimensions au niveau d'un établissement, soit pour assurer un réseau étendu multi-établissement (*Wide Area network*). Le pont, équivalent à une passerelle, intervient au niveau 2 (liaison de données) du modèle ISO (Interconnexion de Système Ouvert). Concrètement, il travail comme un filtre en faisant basculer d'un réseau sur l'autre les trames dont l'adresse ne figure pas dans le premier réseau. Cette technique est employée, par exemple, pour raccorder deux câbles formant chacun un réseau Ethernet afin de leur permettre de communiquer entre eux.

Port (*Port*)
Interface physique entre une ligne et un équipement. En général, le port est matérialisé par une prise montée sur une carte du micro ou du modem. Le port est géré par un "driver" de communication. Pour les télécommunications, il peut servir à la connexion entre un ETTD (Equipement Terminal de Transmission de Données) et un ETCD (Equipement de Terminaison de Circuit de Données).

Présélection
Sélection, par un personnel autorisé, d'événements auditables qui doivent être enregistrés dans des fichiers de trace.

Post-sélection
Sélection, par un personnel autorisé, d'événements spécifiques qui ont été enregistrés dans des fichiers de trace.

Progiciel (*Software Package*)
Ensemble cohérent et indépendant de programmes, de services, d'interfaces utilisateur comportant une documentation, conçus pour réaliser des traitements informatiques standards. Leur diffusion peut alors revêtir un caractère commercial de grande envergure. Un utilisateur peut le mettre en oeuvre et l'employer d'une façon autonome.

Protocole (*protocol*)
Séquence de règles à suivre dans les télécommunications pour établir et entretenir des échanges d'informations entre des entités distantes. Il recouvre des procédures, et s'adresse plutôt à des échanges complexes: on parle de procédure HDLC (*High-level Data Link Control*) mais de protocole X25 ou encore de protocole de transfert de fichier. On distingue plusieurs catégories de protocoles, parmi lesquelles des protocoles d'accès (aux réseau publics); des protocoles d'extrémité; des protocoles de bout en bout; des protocoles de communication, de transmission, de conversion, etc. Un grand nombre de protocoles sont normalisés par des organismes publics comme le CCITT (Comité Consultatif International Télégraphique et Téléphonique), l'ISO (*International Standards organization*), l'ECMA (*European Computer Manufacturers Association*) ou autre. Les plus connus sont publiés par le CCITT sous la forme de recommandations.

Raccordement (*connection*)
Mise en liaison de deux sections de câble. Ce terme est aussi utilisé dans l'expression raccordement d'abonné ou d'utilisateur, pour désigner les opérations consistant à relier des équipements au(x) réseau(x) ou au service de communications.

Répéteur (*repeater*)
Equipement placé sur une liaison, pour amplifier ou régénérer les signaux transmis et donc compenser l'affaiblissement ou la distorsion de la liaison. Pour les RLE (Réseaux Locaux d'Entreprise), le répéteur sert à étendre leur portée. Il opère au niveau de la couche 1 (liaison physique) du modèle ISO (Interconnexion des Systèmes Ouverts).

Reroutage (*rerouting*)
Le reroutage permet à un serveur de transférer un utilisateur ou une information vers un autre serveur, sans que cet utilisateur soit obligé de se déconnecter du premier serveur.

Réseau (*network*)
Ensemble d'équipements et de ressources de transmission mis en commun pour offrir des services de communication. Un réseau peut être local ou étendu, national ou international, privé ou public.

Retransmission
Technique par laquelle des informations qui arrivent dans un équipement de commutation, de concentration, ou dans un serveur de communication sont placées en mémoire puis retransmises.

RS 232-C
Interface normalisée américaine, dont l'équivalent au CCITT est V24. Placée entre un micro-ordinateur et un modem qui décrit les caractéristiques physiques d'une prise à 25 "pins" ou broches qui se trouve sur ces deux équipements. Lors de l'installation d'un modem externe, il faut faire attention à la nature des prises: mâle ou femelle, aux deux bouts, et disposer du bon câble.

Satellite
Les satellites de télécommunication ont ouvert le champ à des réseaux hertziens de télécommunication offrant de grandes capacités de sécurité et de bande très large. Ils permettent de s'affranchir des distances et des contraintes physiques du câblage. Les services offerts par des satellites comme TELECOM 1 et 2 restent chers et réservés à des applications de très gros volume.

Une grande partie du trafic téléphonique intercontinental est assuré par satellite. On parle aussi de satellite comme dispositif à la périphérie locale ou distante d'un système informatique assumant les fonctions communes à un ou plusieurs matériels en lieu et place du site central.

Sujet
Une entité active, généralement en forme d'une personne, processus ou dispositif qui fait circuler l'information à travers des objets ou qui change l'état du système.

Telnet
C'est le protocole standard Internet qui offre un service de connexion à distance avec un terminal. Telnet permet à un utilisateur situé dans un site d'interagir avec des systèmes à temps partagé situés dans d'autres sites. C'est l'utilisateur qui invoque le programme d'application Telnet pour se connecter à une machine distante.

Trame (*frame*)
Suite d'information structurées, constituée de caractères de commande et de données qu'un protocole de communication prend en compte, dans un système de communication.

Bibliographie

[Asax 92]
N. Habra, B. Lecharlier, A. Mounji, I. Mathieu, "Preliminary report on **A**dvanced **S**ecurity **A**udit Trail Analysis on uni**X** (ASAX also called SAT-X)", January 15, 1992.

[Asax 93a]
N. Habra, B. Lecharlier, A. Mounji, I. Mathieu, "ASAX: Software Architecture and Rule-Based Langage for Universal Audit Trail Analysis", 1993.

[Asax 93b]
N. Habra, B. Lecharlier, A. Mounji, "ASAX: Specification of some extensions to the Analyzer", February 3, 1993.

[Asax 93c]
N. Habra, B. Lecharlier, A. Mounji, I. Mathieu, "Preliminary report on **A**dvanced **S**ecurity **A**udit Trail Analysis on uni**X** (ASAX also called SAT-X)", Septembre 14, 1993.

[Asax 93d]
N. Habra, B. Lecharlier, A. Mounji, I. Mathieu, "**A**dvanced **S**ecurity **A**udit Trail Analysis on uni**X** (ASAX also called SAT-X) Implementation design of the NADF Evaluator", September 14, 1993.

[Asax 94]
N. Habra, B. Lecharlier, A. Mounji, "ASAX: Specification of some extentions to the Analyser", August 19, 1994.

[Bac 92]
C. Bac, D. Bouillet, "Administrer des systèmes UNIX en réseau", DUNOD informatique, 1992.

[Cerf 83]
V. G. Cerf, E. Cain, "The DoD Internet Architecture Model", Computer Networks, 1983.

[Comer 88]
D. Comer, "Internetworking with TCP/IP", Prentice Hall, 1988.

[Daisomont 88]
JP. Daisomont, "Reflexion sur le modèle TCP/IP, comparaison et analyse d'implémentation", mémoire, 1988.

[Darimont 92]
A. Darimont, "Programmer en langage C", Editions micro application, 1992.

[DoD 83]
Departement of Defense (DoD) Computer Security Center, "Trusted Computer System Evaluation Criteria" (Orange Book), DoD, December 1983.

[Ethernet 82]
Digital, Intel, Xerox, "The Ethernet, a Local Area Network, Data-Link Layer and Physical Layer Specifications", version 2.0, 1982.

[Henshall 90]
J. Henshall, S. Shaw "OSI EXPLAINED End-to-End Computer Communication Standards", Second Edition, ELLIS HORWOOD, 1990.

[Kernighan 92]
B.W. Kernighan, D.M. Ritchie, "Le langage C", 2e édition, Masson, Prentice Hall, 1992.

[Libion 91]
F. Libion, "*Toward 'intelligent Security Audit Trail Analysis Tools*", MS Thesis, FUNDP, Institut d'informatique, Namur, 1990/1991.

[Lunt & Al 89]
T.F. Lunt, R. Jagannathan, R. Lee, A. Whitehurst, S. Listgarten "*Knowledge-based Intrusion Detection*", in Prooceeding of the 1989 AI Systems in Government Conference, Washington, DC, March 1989.

[Nemeth 89]
E. Nemeth, G. Snyder, S. Seebass, "UNIX System Administration Handbook", Prentice Hall, 1989.

[Stallings 85]
W. Stallings, "Data and Computer Communications", Mac millan, New York, 1985.

[Stallings 89]
W. Stallings, "The TCP/IP protocol suite", Sams Carmel, 2nd ed. 1989.

[Tanenbaum 89]
S. Tanenbaum, "COMPUTER NETWORKS", Prentice-Hall International Editions, 1989.

[TCSEC 85]
"*Trusted Computer System Evaluation Criteria*", in "*Orange Book*", Departement of Defense, NCSC, National Computer Center, December 1985.

[Thomas 85]
R. Thomas, J. Yates, "UNIX", Osborne McGraw-Hill, 1985, pp. 45-589.

[RFC 793]
TCP - DARPA INTERNET PROGRAM - PROTOCOL SPECIFICATION. September 1981. Prepared for DARPA Information Processing Techniques Office, by Information Sciences Institute - University of Southern California. Marina del Rey, California

[RFC 791]
IP - DARPA INTERNET PROGRAM - PROTOCOL SPECIFICATION. September 1981. Prepared for DARPA Information Processing Techniques Office, by Information Sciences Institute - University of Southern California. Marina del Rey, California

[Zimmermann 83]
H. Zimmermann, J.D. Day, "The OSI Reference Model", Proceeding of the IEEE, volume 71, number 12, December 1983.

Annexe A.

EXEMPLES DE REGLE RUSSEL

A.1. Exemples de règles RUSSEL.

Les deux premiers exemples se basent sur un fichier de trace standardisé généré par le système d'audit des commandes SunOS qui est totalement différent de celui généré par la commande etherfind. Cependant, la méthode de traitement est la même. Le dernier exemple se base sur un fichier de trace généré par etherfind.

A.1.1. Détection d'une tentative d'intrusion externe.

Les appels de procédures externes seront notés en italique, les mots clés sont notés en gras, "evt", "res", "terminal", et "timestp" identifient les champs de l'enregistrement de données du fichier de trace normalisé.

Le symptôme qui va être détecté est l'occurrence du nombre "maxtime" d'un login qui a échoué pendant une période de "duration" secondes. Pour simplifier tous les logins qui ont échoués vont être considérés quelques soit l'identité de l'utilisateur.

rule Failed login (maxtimes, duration: integer)
This rule detects a first failed login and triggers off an accounting rule with an expiration time

begin
if evt = "login" **and** res = "failure" **and** is_*unsecure* (terminal)
Trigger off for_next Count rule1 (maxtimes-1, timestp + duration)
fi;

Trigger off for_next Failed login (maxtimes , duration)
end

```
rule  Count rule1 (countdown , expiration : integer)
#This rule counts the subsequent failed logins,
#it remains active until its expiration time or until the countdown becomes 0

if evt = "login" and  res = "failure" and is_unsecure(terminal) and timestp < expiration
                if countdown > 1
                                Trigger off for_next  Count rule1(countdown-1, expiration);
                   countdown=1
                                SendMessage ("too much failed login's")
                if ;
   timestp >= expiration
        Skip;
true
        Trigger off for_next   Count rule1  (countdown, expiration)
fi
```

A.1.2. Détection d'une tentative d'intrusion interne.

Le symptôme à détecter consiste en une succession de "*maxtimes*" fois d'un événement douteux qui échoue durant une période de "*duration*" secondes. Un tel événement peut être, par exemple, l'exécution de la commande de changement de privilèges d'accès à un fichier.

Pour cela deux règles sont utilisées. La première règle **Abusive_user** détecte une première commande qui échoue et déclenche la règle **Count_rule2.** Cette dernière compte le nombre de commandes qui échouent pour l'utilisateur suspect. Elle est active jusqu'à expiration du temps spécifié. Le code de cette règle et bien d'autres requêtes sont disponibles dans [Asax 93a].

A.1.3. Affichage du contenu d'un fichier de traces.

Etant donné que les fichiers normalisés des traces sont sous un format binaire, il est souvent utile de disposer d'un module de règles permettant l'affichage du contenu de ce fichier sous une forme lisible. En effet, cela peut aider à contrôler les règles de détection.

Nous pouvons ainsi afficher le contenu de l'enregistrement ayant satisfait à une condition donnée et de vérifier directement qu'il n'y a pas eu d'erreur. Nous pouvons également se baser sur ce format lisible pour construire des règles de détection de scénarios.

Cette règle fait appel à deux autres règles:

1. *SHOW_REC* qui permet d'afficher le contenu de l'enregistrement courant et de se réactiver d'elle-même pour l'enregistrement suivant;

2. *COUNT_REC* permet d'afficher le nombre d'enregistrements du fichier traité. La procédure prédéfinie *display_current* reçoit l'adresse de l'enregistrement courant et affiche son contenu sous le format suivant:

nom_du_champ [identifiant longueur] = valeur

Cette règle est implémentée dans le cadre de ce mémoire. Au chapitre III.2. nous avons présenté les spécifications de cette règle ainsi que d'autres requêtes simples. Ces requêtes peuvent être utilisées avec différentes combinaisons de filtrage. La reconstitution des données filtrées sera accomplie par l'intermédiaire d'une routine C qui sera appelée par la requête.

Le code de la règle *Show_rec* est illustré ci-dessous:

```
global v 1 : integer;
rule SHOW_REC;
begin
        print ("=========================================");
        display_current;
        v1 := v1 + 1;
        trigger off for_next SHOW_REC
end;

rule COUNT_REC;
begin
    println (' ') ;
    println ("================");
    print  (v1);
    println (" enregistrements(s)");
    println ("================");
    println (")
end;

init_action;
begin
  v1 := 0;
  trigger off for_next SHOW_REC;
  trigger off at_completion COUNT_REC
end.
```

Nous pouvons également faire des filtrages variés sur les paquets à analyser en ne gardant que certains enregistrements d'audit des événements pertinents pour l'analyse. Nous pouvons, par exemple, sélectionner tous les événements survenus sur une machine particulière et dans un port spécifique sur cette machine. Nous pouvons également surveiller le réseau en visualisant tous les paquets captés pendant une période chargée de la journée etc. En plus du filtrage au niveau de la commande etherfind (cf. Annexe C.), il existe plusieurs autres combinaisons de filtrage des paquets captés. Ces filtrages se font au niveau des données d'audit, ce qui explique la puissance de l'outil ASAX et de son langage RUSSEL.

Dans ce module, les enregistrements satisfaisants à une certaine condition de présélection (ou de filtrage) sont affichés sur l'écran, les autres sont simplement ignorés. Une variable globale est utilisée pour compter le nombre d'enregistrements filtrés. La règle d'initialisation *init-action* permet d'ouvrir le fichier de sélection pour l'enregistrement suivant et d'activer la règle de sélection pour l'enregistrement qui fermera le fichier de sélection et affichera le nombre de records sélectionnés.

L'appel des routines C est possible avec le langage RUSSEL, ainsi, la requête convertie les données lues par la routine "traduit (data, donnee)" qui est implémentée à part et qui est intégrée au système ASAX par des interfaces spécifiques. Le code de cette requête est illustré ci-deesous:

```
global v1: integer;
rule affich_enreg;
begin
        if (IPS %= '138.48.5.57' and IPD %= '138.48.4.10' and DP %= '1023')
            → begin
                        println ("==================================");
                        if (present data = '1')
                        → begin
                              traduit (data, donnee);
                              println ('La donnee = ', data)
                           end;
                        true → println ('La donnee = ', Data)
                        fi;
                v := v + 1;
                println ('IPS = ', IPS , 'IPD = ', IPD);
                println ("==================================");
                end
        fi;
        trigger off for_next affich_enreg
end;

rule compt_enreg;
begin
        println ("==================================");
        print   (v1);
        println (" enregistrement(s)  ");
        println ("==================================")
end;

init_action;
begin
        v1 := 0;
        trigger off for_next affich_enreg;
        trigger off at_completion compt_enreg
end.
```

Annexe B.

LA SYNTAXE ET LA SEMANTIQUE DU LANGAGE RUSSEL

B.1. Rappel: Bref aperçu sur le langage RUSSEL.

RUSSEL est un langage de règles permettant d'analyser tout fichier séquentiel structuré, et en particulier les fichiers NADF générés par l'adaptateur de format (cf. III.1.).

Une règle est considérée comme une procédure, au sens classique, avec des paramètres et des variables locales. Les paramètres permettent le passage d'informations entre les appels successifs des règles et donc entre les enregistrements.

Trois types de règles sont à distinguer:

1. des règles d'initialisation: elles sont activées au début du traitement, pour le premier enregistrement. Leur but est d'initialiser le processeur d'analyse en activant des règles pour le premier enregistrement;

2. des règles de sélection: ce sont des règles actives pendant l'évaluation d'un enregistrement donné. A chaque instant, cet ensemble de règles peut être modifié;

3. des règles de terminaison: elles sont évaluées après la fin du traitement du fichier de traces. Leur déclenchement peut activer, par exemple, l'impression de rapports sur les résultats de l'analyse, de la fermeture des fichiers, etc.

Une règle sert à détecter un événement dans un scénario, elle renferme un ensemble de conditions et une séquence d'actions. Les actions se présentent sous deux formes:

- des actions spéciales basées sur un mécanisme de déclenchement de règles (*trigger off*), le déclenchement d'une nouvelle règle consiste à donner son nom, les valeurs des paramètres et un paramètre spécial indiquant si la règle sera activée pour le record courant (**trigger off for_current**), le record suivant (**trigger of for_next**) ou la fin de l'analyse (**trigger off at_completion**). Ces modes de déclenchement seront développés dans le paragraphe A.4.

Après avoir évalué un enregistrement, la règle est automatiquement désactivée, à moins d'être réactivée explicitement par elle-même ou à l'aide d'une autre règle;

- des actions classiques sous forme d'appels de procédure prédéfinie permettant d'implémenter des fonctions de haut niveau (envoyer un message, déclencher une alarme, générer des fichiers temporaires, etc.).

Nous pouvons utiliser les règles pour détecter des tentatives d'intrusion et de violation de sécurité, ou pour faire des statistiques et des contrôles sur le taux d'utilisation des ports, des machines, etc. On peut supposer, pour une violation ou une infraction quelconque, un symptôme simplifié. Un ensemble de règles RUSSEL peut ainsi être proposé pour détecter ce symptôme. Ceci se fait en analysant le fichier NADF des traces. Les requêtes feront des filtrages multiples et variés pour detecter les informations contenues dans les messages envoyés sur le réseau.

L'objectif n'étant pas de développer toutes les sélections possibles, complète et sophistiquée mais plutôt de montrer la façon dont RUSSEL analyse ces informations (bytes captés) en filtrant sur les champs pertinents.

B.2. Types de donnée utilisées.

Généralement deux types de donnée sont considérés:

- string de bytes;
- entier.

Toute donnée peut être représentée par un string de bytes. Les types de donnée doivent être le plus simple possible pour pouvoir assurer une compatibilité avec la grande variété de formats des fichiers de trace. ASAX utilise principalement le string de bytes; le type de donnée entier est parfois utilisé pour des opérations arithmétiques nécessaires pour spécifier des opérations de sélection complexes sur les "*data records*"

B.3. La syntaxe du langage RUSSEL.

B.3.1. Les items lexicaux.

Définition des items lexicaux ou tokens sous forme BNF:

<token > ::= <identifier > | <constant > | <special symbol >
<identifier > ::= <letter >
| <identifier > <digit >
| <identifier > <letter >
| <identifier > <underscore > <letter >
| <identifier > <underscore > <digit >

<constant > ::= <integer constant >
| <C_literal >
| <X_literal >

<integer constant > ::= <digit >
| <integer constant > <digit >

<C_literal > ::= '<C_sequence >'
<C_sequence > ::= <empty >
| <C_sequence > <C_character >

<C_character > ::= any printable ASCII character except simple quote (')|"
<X_literal > ::= X '<X_sequence >'

<X_sequence > ::= <empty > | <X_sequence > <X_symbol >
<X_symbol > ::= <X_character > <X_character >
<X_character > ::= 0 | ... | 9 | A | B | C | D | E | F

<digit > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<letter > ::= a | ... | z | A | ... | Z

<special symbol > ::= + | - | * | (|) | > | < | != | <= | >= | : | %= | ; | = | -> | := | , | and
| at_completion | begin | div | do | end | false | fi | for_current | for_next | not
| od | off | or | present | rule | skip | string | if | integer | mod | trigger | true | var

Le programme ASAX est constitué d'une séquence finie d'items lexicaux. Cette séquence doit respecter les règles syntaxiques suivantes:

B.3.2. La syntaxe abstraite.

La syntaxe abstraite du langage RUSSEL est la suivante:

<rule declaration > ::= <rule heading > <variable declaration part > <action part >
<rule heading > ::= **rule** <rule name > (<parameter list >)
<rule name > ::= <identifier >

<parameter list > ::= <empty >
| <parameter group > ; ... <parameter group >

<parameter group > ::= <parameter name >, ... , <parameter name > : <type >
<parameter name > ::= <identifier >

<type > ::= **ascii_string** | **integer**

<variable declaration part > ::= <empty >
| **var** <variable declaration >; ... ; <variable declaration >

<variable declaration > ::= <variable name >, ... , <variable name > : <type >

<variable name > ::= <identifier >
<action part > ::= <action >

<action > ::= **skip**
| <assignment >
| <conditional action >
| <repetitive action >

| <compound action >
| <rule triggering >
| <predefined procedure call >
<assignment > ::= left expression > := <right expression >

<left expression > ::= <parameter name > | <variable name >
<right expression > ::= <expression >

<expression > ::= <constant >
| <field name >
| <left expression >
| <expression > <binary arithmetic operator > <expression >
| <unary arithmetic operator > <expression >

<field name > ::= <identifier >
<conditional action > ::= **if** <guarded action >; ... ; <guarded action > **fi**

<guarded action > ::= <condition > → <action >

<condition > ::= **true** | **false** | **present** <field name>
| <expression > <relational operator > <expression >
| <condition > <logical operator > <condition >
| **not** <condition >

<relational operator > ::= < | > | ≠ | = | <= | >=
<repetitive action > ::= **do** <guarded action > ; ... ; <guarded action > **od**

<compound action > ::= **begin** <action > ; ... ; <action > **end**
<rule triggering > ::= **trigger off** <triggering mode > <rule call >

<triggering mode > ::= **for_next**
| **for_current**
| **at_completion**

<predefined procedure call > ::= <predefined procedure name >
(<expression >, ... ,<expression >)

<predefined procedure name > ::= <identifier >
<rule call > ::= <rule name > (<expression > , ... , <expression >)
<rule name > ::= <identifier >

La correction syntaxique complète d'une expression, d'une condition, ou d'une construction dépend de la table d'identifiants qui regroupe les noms de paramètres, les noms de variables et les noms des champs. Cette table s'appelle *identifier table*.

B.3.3. La syntaxe concrète.

La syntaxe concrète du langage RUSSEL est la suivante:

<condition > ::= <disjunction >
<disjunction > ::= <conjunction > | <disjunction > **or** <conjunction>
<conjunction > ::= <simple condition > | <conjunction > **and** <simple condition >

<simple condition > ::= **true** | **false** | **present** <field name >
| <relational expression > | (<condition >)
| **not** <condition >

<relational expression > ::=
<arithmetic expression > <relational operator > <arithmetic expression >

<relational operator > ::= < | > | ≠ | = | <= | >=
<arithmetic expression > ::= <term >
| <arithmetic expression > <additive operator > <term >

<term > ::= <factor > | <term > <multiplicative operator > <factor >
<factor > ::= <simple expression >

<simple expression > ::= <parameter name >
| <variable name >
| <integer_constant >
| (<arithmetic expression >)
| <pre-defined procedure call >

<multiplicative operator > ::= * | **div** | **mod**
<additive operator > ::= + | -

La priorité des opérateurs de cette syntaxe concrète est:

Priority	*Operators*
1	***, mod, div**
2	+ , -
3	<, >, <=, >=, =, <>
4	**not, present**
5	**and**
6	**or**

La priorité des opérateurs est implicite à la syntaxe concrète. Ces niveaux de priorité sont choisis de façon à ce que les expressions puissent être évaluées normalement.

B.4. La sémantique du langage RUSSEL.

Un petit rappel sommaire sur la sémantique RUSSEL serait utile pour pouvoir comprendre les exemples de règles présentées dans cette Annexe. Pour de plus amples détails, le lecteur peut consulter [Habra 93a] et [Habra 93b].

Nous allons passer en revue les 5 points essentiels suivants:
1. l'évaluation des expressions;
2. l'exécution des actions;
3. le déclenchement des règles;
4. l'exécution d'un appel de procédure prédéfinie;
5. le mécanisme du traitement et l'algorithme général de traitement.

Puis nous aborderont les concepts de base suivants, particuliers à ce langage:

1. *Current Record (CR)*: c'est l'enregistrement courant du fichier NADF, il se compose des champs suivants:
- identifiant;
- longueur;
- valeur.

2. *Current Environment (CE)*: l'environnement courant est un ensemble d'informations décrivant l'état d'exécution d'un programme. Il consiste en:
- l'enregsitrement courant;
- l'environnement local: c'est l'ensemble des variables telles que les variables globales, les variables locales et les paramètres;
- *DStrig*: l'ensemble des règles actives ou déclenchées pour l'enregistrement courant;
- *DSnext*: l'ensemble des règles actives pour l'enregistrement suivant;
- *DScompl*: l'ensemble des règles actives après l'analyse de tout le fichier de trace NADF.

B.4.1. L'évaluation des expressions.

L'évaluation des expression est de quatre types:
1. Une constante littérale (C-littéral ou X-littéral) est évaluée au string du bytes qu'elle représente.

2. Une expression de la forme "**present** *fieldname*" est dévaluée à la valeur VRAI si l'enrgistrement courant contient le champ *fieldname*, sinon elle est évaluée à FAUX.

3. L'évaluation des expressions arithmétiques et relationnelles est la même que pour les autres langages de programmation connus.

4. De même pour les expressions conditionnelles simples et composées.

B.4.2. L'exécution des actions.

Cinq types d'exécutions d'actions sont disponibles:
1. La construction **skip**: elle correspond à l'action vide. Son utilisation permet de clarifier le code d'une règle.

2. L'assignation: l'exécution de l'action *lexpr:= rexpr* par rapport à l'environnement courant revient à, d'abord, évaluer l'expression *rexpr* par rapport à l'environnement courant, puis affecter la valeur trouvé à la variable *lexpr*.

3. L'action conditionnelle: c'est l'exécution, par rapport à l'environnement courant, de l'action suivante:

if
cond 1 → action 1;
...
cond n → action n
fi

La première condition est évaluée; si elle est vrai, l'action "*action 1*" est exécutée par rapport à l'environnement courant et l'exécution est terminée; Sinon:
a) si n > 1 l'exécution reprend à:

if
cond 2 → action 2;
...
cond n → action n
fi

b) sinon l'exécution est terminé.

4. L'action répétitives: l'exécution de l'action répétitive, par rapport à l'environnement courant, est effectuée de la façon suivante:

do
cond 1 → action 1;
...
cond n → action n
od

Les conditions *cond 1,..., cond n* sont successivement évaluées par rapport à l'environnement courant jusqu'à ce que l'une d'entre elles soit évaluée à VRAI. Soit *cond i* cette condition. Dans ce cas, l'action *action i* est exécutée et ensuite l'action répétitive de départ est exécutée à nouveau; si aucune des conditions n'est évaluée à VRAI, l'exécution de l'action répétitive est terminée.

5. Les actions composées: l'exécution, par rapport à l'environnement courant, de l'action:

begin *action 1;...;action n* **end**

consiste à exécuter successivement les *actions i* par rapport à cet environnement.

B.4.3. Le déclenchement de règles.

Soit E un environnement courant et *rule_name* une règle. L'exécution de l'action: **trigger off** *tr_mode rule_name (expr 1,... , expr n);* (où *n* > 0, et *tr_mode* (*triggering mode*) signifie mode de déclenchement) est effectué comme suit:

1. Les expressions *expr 1, ..., expr n* sont évaluées par rapport à l'environnement courant. Soit L = v 1,... , v n = la liste de leurs valeurs respectives;

2. L'effet de l'exécution est d'ajouter cette règle à l'ensemble des règles actives:

- pour le record courant si *tr_mode* = **for_current;**
- pour le record suivant si *tr_mode* = **for_next;**
- à la fin de l'analyse si *tr_mode* = **at_completion.**

B.4.4. L'exécution d'un appel de procédures prédéfinies.

Soit *proc_name* une procédure prédéfinie, E l'environnement courant. La syntaxe de l'exécution de l'appel de procédure prédéfinie est la suivante: *proc_name* (expr 1,... expr n), avec n >= 0. Pendant l'exécution, on évalue d'abord les expressions expr 1,... , expr n, par rapport à E, puis on passe leurs valeurs respectives à la procédure: *proc_name* (v 1,... ,v n). Cette procédure est enfin exécutée par rapport à E (environnement courant).

B.4.5. Le mécanisme de traitement et l'algorithme général de traitement.

a) l'environnement initial

Supposons qu'on dispose, dans l'étape d'initialisation, des n règles actives suivantes: RI_1, ..., RI_n, et, à la fin, des p règles actives suivantes: RC_1, ..., RC_p . Supposons, également, que nous avons la règle suivante:

rule init_rule;
begin
    Trigger off for_next RI_1;
    ...
    Trigger off for_next RI_n;
    Trigger off at_completion RC_1;
    ...
    Trigger off at_completion RC_p;
end;

L'environnement initiale correspondant à ces règles d'initialisation et de fin de traitement est formé. Les composants suivants constituent cet environnement:

- le premier enregistrement du fichier de traces;
- l'environnement local (vide) de l'environnement initial;
- DStrig = { *init_rule* };
- DSnext = { };
- DScompl = { }.

Il est à noter que:

1) L'environnement initial devrait comprendre une description des procédures prédéfinies contenues dans la librairie ainsi que la description de quelques variables globales telle que la table du format adaptateur. Cependant, pour des raisons de simplicité, ces composants de l'environnement initial sont faits implicitement.
2) L'environnement initial devrait être définie comme suit:

- le premier enregistrement du fichier de traces;
- un environnement local vide;
- DStrig = { RI_1, ..., RI_n };
- DSnext = { };
- DScompl = { RC_1, ..., RC_p }.

b) l'algorithme général de traitement.

Les *inputs* de cet algorithme sont:

- le fichier de traces sous forme NADF;
- un ensemble de définitions de règles composé de trois parties:
  1. DSinit: ensemble de définitions de règles d'initialisation;
  2. DStrig: ensemble de définitions de règles, correspondant à des règles devant être déclenchées dynamiquement;
  3. DScompl: ensemble de définitions de règles de fin de traitement.

Les *outputs* sont: tous types de rapports générés pendant le traitement.

Au début de l'algorithme, DStrig est vide. La règle d'initialisation est formée des éléments de DSinit et DScompl.

Algorithme.

Initialisation:
- Scurr:= init_rule;
- Scompl:= { };
- Ouvrir le fichier de traces;

En effet, au départ, la seule règle active est la règle de nom réservée **init_rule** alors que l'ensemble des règles actives à la fin de l'analyse est vide.

condition de fin de boucle:
- fin du fichier de traces NADF

Itération:
- lire l'enregistrement suivant;
- traiter Scurr (en analysant l'enregistrement courant) pour produire NSnext et NScompl;
- Scurr:= NSnext;
- Scomp:= NScompl ∪ Scopml;

En général, l'exécution des règles actives pour l'enregistrement courant peut activer d'autres règles pour l'enregistrement suivant et peut aussi activer d'autres règles pour la fin de l'analyse. Ces derniers viennent s'ajouter à l'ensemble des règles actives à la fin.

Clôture:
- fermer le fichier de traces NADF;
- exécuter les règles actives de fin de traitement

Annexe C.

DESCRIPTION DE LA COMMANDE ETHERFIND

C.1. La présentation de la commande.

La commande etherfind est une commande de maintenance, elle capte les paquets sur le support Ethernet. Elle est disponible dans les options installées avec le logiciel du réseau. Sa syntaxe est la suivante [man etherfind, Sun Release 4.1]:

etherfind [-d] [-n] [-p] [-r] [-t] [-u] [-v] [-x] [-c count] [-i interface] [-l length] expression .

C.2. Sa description.

Etherfind imprime les informations qu'elle capte sur le réseau. Ces informations sont regroupées en paquets (cf. Partie II.) et circulent sur le support Ethernet. Ils sont ceux qui correspondent à l'expression booléenne expression. Un affichage court, sans l'option -v, affiche seulement la destination et la source (avec les numéros des ports). Quand un paquet Internet est fragmenté en plusieurs paquets Ethernet, tous les fragments excepté le premier sont marqués d'un astérisque. Avec l'option -v, l'affichage est plus verbeux, on obtient une trace qui pourrait nous aider à analyser plusieurs problèmes du réseau.

Selon l'option choisi, on peut capter différentes informations; les plus pertinentes se trouvent dans la zone de donnée du paquet analysé. L'option qui nous permette de capter ces données est l'option [-x]. Cette option permet d'ajouter dans le fichier de traces les informations qu'elle vient de capter. Ces informations seront écrites en hexadécimal. Cette option capte donc tous les paquets qui sont émis ou reçus d'une machine ou entre deux machines; les adresses de ces machines sont spécifiées dans expression.

C.3. Ses options.

-d: Imprime le nombre de paquets perdus. Pas nécessairement fiable.

-n: Ne convertit pas en noms les adresses des hôtes et les numéros de port.

-p: L'interface ne sera pas pris au hasard.

-r: Mode RPC: traite chaque paquet comme étant un message RPC, imprimant ainsi les numéros de programme et de procédure. Les paquets routés sont également décodés de façon plus complète, et les arguments du NIS (*Network Information Service*) et de NFS (*Network File System*) seront imprimés.

-t: Le *Timestamps*: précède chaque paquet avec la valeur du temps en secondes et en centième de secondes à partir du premier paquet.

-u: Met les lignes de sortie dans un buffer.

-v: Le mode verbeux: imprime quelques champs des paquets TCP et UDP (cf. Partie II.).

-x: Transforme les paquets en hexadécimal, en plus de l'impression, par défaut, des lignes pour chaque paquet. L'option -l limite cette impression.

-c count:
Capte seulement count paquets avant de sortir. Ce qui est parfois utile de déposer un échantillon du trafic Ethernet dans un fichier pour une analyse ultérieure.

-i interface:
Etherfind écoute sur l'interface. Le programme netstat (8C) avec l'option -i liste tous les interfaces dont dispose la machine.

-l length:
Utilisée avec l'option -x pour limiter le nombre de bytes imprimé.

C.4. Ses expressions.

dst destination:
Vraie si le champ de destination du paquet est destination, qui peut être une adresse ou un nom.

src source:
Vraie si le champ source du paquet est source, qui peut être une adresse ou un nom.

host name:
Vraie si la source ou la destination du paquet est nom.

between host1 host2:
Vraie si la source du paquet est host1 et la destination est host2, ou la source est host2 et la destination est host1.

dstnet destination:
Vraie si le champ de destination du paquet a une partie réseau de destination, qui peut être une adresse ou un nom.

srcnet source:
Vraie si le champ source du paquet a une partie réseau source, qui peut être une adresse ou un nom.

srcport port:
Vraie si le paquet a un port source de valeur port. Ceci est pour vérifier la valeur du port source des paquets TCP ou UDP (cf. Partie II.). Le port peut être un nombre ou un nom utilisé dans /etc/services.

dstport port:
Vraie si le paquet a un port de destination dont la valeur est port. Le port peut être un nombre ou un nom.

less length:
Vraie si le paquet a une longueur inférieure ou égale à length.

greater length:
Vraie si le paquet a une longueur supérieure ou égale à length.

-proto protocol:
Vraie si le paquet est un paquet IP (cf. Partie II.) de type de protocole protocol. Protocol peut être un nombre ou un des noms suivants icmp, udp, nd, ou tcp.

byte byte op value:
Vraie si le byte numéro byte du paquet a une relation op avec value. Les valeurs que peut prendre op sont: +, <, >, &, et | . Ainsi 4=6 est vraie si le quatrième byte du paquet a la valeur 6, et 20&0xf est vraie si le vingtième byte a un de ses quatre bits de bas ordre non nul.

broadcast:
Vraie si le paquet est un paquet de diffusion.

arp:
Vraie si le paquet est un paquet ARP.

rarp:
Vraie si le paquet est un paquet RARP.

-ip:
Vraie si le paquet est un paquet IP.

-decnet:
Vraie si le paquet est un paquet DECNET.

-apple:
Vraie si le paquet est un paquet du protocole AppleTalk.

Les premières expressions, qui sont les plus importantes, peuvent être combinées en utilisant les opérateurs suivants:
- les parenthèses;
- la négation ('not');
- la concaténation ('and');
- l'alternance ('or').

Annexe D.

EXEMPLE DE FICHIER DE TRACES

La commande etherfind permet de capter des paquets sur le réseau. Ces paquets peuvent être stockés dans des fichiers de traces. Ces fichiers ont un format particulier, ils sont de type ascii. Le fichier ci-dessous est généré par la commande suivante:
etherfind -x -c 10 src celeri ↵. Qui veut dire: capter (etherfind), sous forme d'héxadécimale (-x), 10 paquets (-c 10), ayant comme source la machine celeri (src celeri). Le fichier sera afficher à l'écran, il peut être également rediriger vers un fichier.

```
Using interface le0

                                                    icmp type
lnth   proto          source        destination     src port    dst port
 60     tcp           celeri           salade         1023       login
08 00 20 0a 4c 40 08 00 20 0a 9c 23 08 00 45 00
00 29 db b0 00 00 3c 06 84 51 8a 30 05 39 8a 30
05 34 03 ff 02 01 2d 97 08 be 78 9b c2 f0 50 18
10 00 9c 1c 00 00 6d 9b 50 18 10 00

 60     tcp           celeri           salade         1023       login
08 00 20 0a 4c 40 08 00 20 0a 9c 23 08 00 45 00
00 28 db b1 00 00 3c 06 84 51 8a 30 05 39 8a 30
05 34 03 ff 02 01 2d 97 08 bf 78 9b c2 f1 50 10
10 00 09 24 00 00 6d 61 70 6f 6e 2e

 60     tcp           celeri           salade         1023       login
08 00 20 0a 4c 40 08 00 20 0a 9c 23 08 00 45 00
00 29 db b2 00 00 3c 06 84 4f 8a 30 05 39 8a 30
05 34 03 ff 02 01 2d 97 08 bf 78 9b c2 f1 50 18
10 00 9a 1a 00 00 6f f0 50 18 10 00

 60     tcp           celeri           salade         1023       login
08 00 20 0a 4c 40 08 00 20 0a 9c 23 08 00 45 00
00 28 db b3 00 00 3c 06 84 4f 8a 30 05 39 8a 30
05 34 03 ff 02 01 2d 97 08 c0 78 9b c2 f2 50 10
10 00 09 22 00 00 6f bf 50 18 10 00

 60     tcp           celeri           salade         1023       login
08 00 20 0a 4c 40 08 00 20 0a 9c 23 08 00 45 00
00 29 db b4 00 00 3c 06 84 4d 8a 30 05 39 8a 30
```

05 34 03 ff 02 01 2d 97 08 c0 78 9b c2 f2 50 18
10 00 97 18 00 00 72 f1 50 18 10 00

60 tcp celeri salade 1023 login
08 00 20 0a 4c 40 08 00 20 0a 9c 23 08 00 45 00
00 28 db b5 00 00 3c 06 84 4d 8a 30 05 39 8a 30
05 34 03 ff 02 01 2d 97 08 c1 78 9b c2 f3 50 10
10 00 09 20 00 00 72 c0 50 18 10 00

146 udp celeri patate 1022 2049
08 00 20 0a 75 41 08 00 20 0a 9c 23 08 00 45 00
00 84 db b6 00 00 ff 11 c2 0e 8a 30 05 39 8a 30
04 0a 03 fe 08 01 00 70 00 00 00 0b 16 10 00 00
00 00 00 00 00 02 00 01 86 a3 00 00 00 02 00 00
00 01 00 00 00 01 00 00 00 20 2e 36 84 29 00 00
00 06 63 65 6c 65 72 69 00 01 00 00 02 30 00 00
00 65 00 00 00 01 00 00 00 65 00 00 00 00 00 00
00 00 00 00 07 06 00 00 00 01 00 0a 00 00 00 00
00 02 5e 9f 8c fd 00 0a 00 00 00 00 00 02 5e 9f
8c fd

146 udp celeri patate 1022 2049
08 00 20 0a 75 41 08 00 20 0a 9c 23 08 00 45 00
00 84 db b7 00 00 ff 11 c2 0d 8a 30 05 39 8a 30
04 0a 03 fe 08 01 00 70 00 00 00 0b 16 11 00 00
00 00 00 00 00 02 00 01 86 a3 00 00 00 02 00 00
00 05 00 00 00 01 00 00 00 20 2e 36 84 29 00 00
00 06 63 65 6c 65 72 69 00 01 00 00 02 30 00 00
00 65 00 00 00 01 00 00 00 65 00 00 00 00 00 00
00 00 00 00 07 06 00 00 00 01 00 0a 00 00 00 00
00 0e 17 09 1f 6d 00 0a 00 00 00 00 00 02 5e 9f
8c fd

60 tcp celeri salade 1023 login
08 00 20 0a 4c 40 08 00 20 0a 9c 23 08 00 45 00
00 29 db b8 00 00 3c 06 84 49 8a 30 05 39 8a 30
05 34 03 ff 02 01 2d 97 08 c1 78 9b c2 f3 50 18
10 00 01 17 00 00 08 f2 50 18 10 00

60 tcp celeri salade 1023 login
08 00 20 0a 4c 40 08 20 20 0a 9c 23 08 00 45 00
00 28 db b9 00 00 3c 06 84 49 8a 30 05 39 8a 30
05 34 03 ff 02 01 2d 97 08 c2 78 9b c2 f7 50 10
10 00 09 1b 00 00 00 05 00 00 00 01

Annexe E.

PROGRAMME
DE
L'ADAPTATEUR DE FORMAT

```
/*************************************************/
/*                                               */
/*              L'adaptateur de format           */
/*                                               */
/*************************************************/

/*************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/* */

char c;
char lec[10];
char s[15];
char tb[15];
char hex[15];
char v[1500];
char aux[25];
char dat[25];
char cod[25];
char tab[25];

FILE *fd2;

/* */

int n;
int l;
int tl;
int d;

struct enreg
  {
    unsigned short int n;
    unsigned short int l;
    char v[1500];
    struct enreg *next;
  }  ;

char *ptr_rec;

conv_adr_ip ();
lire_n_hexa ();
int conv_hex ();
long trt_udp();
long trt_tcp ();
ecrire_buf ();
trt_paquet ();

/*************************************************/
/*************************************************/
main(argc, argv)
int argc;
int *argv[];
{
int fd1;
```

```
int rc1;

char  *malloc();


int trouve = 0;
int bool = 0;
char c;
char mot[15];
char l[4];
extern char lec[];
extern char tab[];

/* */

ptr_rec = malloc (4096);

                /* L'ouverture du fichier de traces qui va etre converti en un
                   fichier de format NADF  */

fd2 = fopen (argv[1], "r");
printf("fd2 = %d\n", fd2);
if (fd2 == 0)
  {
    printf("%s : IMPOSS D'OUVRIR %s\n", argv[0], argv[1]);
    exit(1);
  }


                /* Appel de la fonction predefinie creat_NADF pour la creation
                   du fichier NAFDF */

fd1 = creat_NADF (argv[2]);
printf("fd1 = %d\n", fd1);
if(fd1 < 0)
  {
    printf("%s : IMPOSS DE CREER %s\n", argv[0], argv[2]);
    exit(1);
  }

strcpy(l,"");

c = fscanf (fd2, "%s", mot);
while (c != EOF) {

                /* Positionnement sur les deux premiers hexa du premier paquet
                   du fichier des traces  */

  if ((bool != 1) && (strlen(mot) == 2)
      && (((mot[0] <= '9') && (mot[0]>='0')
           || (mot[0] <= 'f') && (mot[0] >= 'a'))
          && ((mot[1] <= '9') && (mot[1]>='0')
              || (mot[1] <= 'f') && (mot[1] >= 'a'))))
    {
      strcpy (l, mot);
      bool = 1;
    }
  else {
    if ((bool == 1) && (strlen(mot) == 2) && (c != EOF)
        && (((mot[0] <= '9')
             && (mot[0] >='0')|| (mot[0] <= 'f')
```

```
            && (mot[0] >= 'a'))&& ((mot[1] <= '9')
                                  && (mot[1]>='0')|| (mot[1] <= 'f')
                                  && (mot[1] >= 'a'))))
      trouve = 1;
    else (bool = 0);
  }
  if (trouve == 1){

                /* Traitement du paquet */

    trt_paquet(fd1, l, mot);

    do {
      c = fscanf(fd2, "%s", mot);
    } while ((strlen(mot) == 2) && (c != EOF));

    trouve = 0;
    bool = 0;
  }
  c = fscanf (fd2, "%s", mot);
}

fclose (fd2);
rc1 = close_NADF (fd1);
printf("rc1 = %d\n", rc1);
free (ptr_rec);
}
/************************************************/
/************************************************/

struct enreg *mallocEnreg()
{
/* Allocation de la memoire pour la creation d'une cellule de la chaine */

        char *malloc();
        struct enreg *ptr;

        ptr = (struct enreg *) malloc (sizeof(struct enreg)  );
        ptr->n = 0;
        ptr->l = 0;
        strcpy(ptr->v, "");
        return(ptr);
}
/************************************************/

lire_n_hexa (d, tb)
int d;
char tb[];
{
/* construit le tableau tb avec les d hexadecimaux qu'elle lit */

int i;
char m[25];

strcpy (tb, "");
for (i = 0;i < d;i++){
    fscanf (fd2, "%s", m);
    strcat (tb, m);
    strcat (tb, " ");
}
}
```

```
/*************************************************/

int conv_hex (hex)
char hex[];
{
/* Convertion, recoit un hexadecimal et retourne un entier */

int val;

if ((hex[0] <= '9') && (hex[0] >= '0'))
   val = hex[0] - '0';

else if ((hex[0] <='f') && (hex[0] >= 'a'))
        val = hex[0] - 'a' + 10;

else if ((hex[0] <='F') && (hex[0] >= 'A'))
        val = hex[0] - 'A' + 10;
else {
        printf("non hexa string %s\n", hex);
        return 0;
}

if ((hex[1] <= '9') && (hex[1] >= '0'))
   val = val * 16 + hex[1] - '0';

else if ((hex[1] <= 'f') && (hex[1] >= 'a'))
        val = val * 16 + hex[1] - 'a' + 10;
else if ((hex[1] <='F') && (hex[1] >= 'A'))
   val = val * 16 + hex[1] - 'A' + 10;
else {
        printf("non hexa string %s\n", hex);
        return 0;
}

return (val);
}
/*************************************************/

conv_adr_ip (n, aux)
int n;
char aux[];
{
/* recoit un hexa, le convertit, le met sous forme d'entiers dans le tableau
   aux, ces entiers sont separes par des points */

int i;
int IP[4];
char tb[3];
char tab[5];

strcpy (aux, "");

for (i = 0;i < n;i++){
  lire_n_hexa (1, tb);
  IP[i] = conv_hex (tb);
}

for (i = 0;i < 4;i++){
  sprintf (tab, "%d", IP[i]);
  strcat (aux, tab);
  if (i < 3)
```

```
    strcat (aux, ".");
}
strcat (aux, " ");
}
/************************************************/

long trt_udp(fin)
struct enreg *fin;
{
/* lit dans le fichier de traces, traite l'information lue, qui est le paquet ud
p */

int  port;
int i;
int len;
int ld;
int lt = 0;
char tab[4];
char inter[1000];
char t[1000];
char don_cod[1000];

struct enreg *ptr_courant, *mallocEnreg();

/* UDP */
/*******/
/*******/

/* SP */

ptr_courant =mallocEnreg();
ptr_courant->n = 316;
lire_n_hexa (1, tab);
port = conv_hex (tab);
lire_n_hexa (1, tab);
port = port * 256 + conv_hex (tab);
sprintf (ptr_courant->v, "%d", port);
ptr_courant->l = strlen (ptr_courant->v);
lt = lt + 4 + ptr_courant->l;

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;


/* DP */

ptr_courant =mallocEnreg();
ptr_courant->n = 317;
lire_n_hexa (1, tab);
port = conv_hex (tab);
lire_n_hexa (1, tab);
port = port * 256 + conv_hex (tab);
sprintf (ptr_courant->v, "%d", port);
ptr_courant->l = strlen (ptr_courant->v);
lt = lt + 4 + ptr_courant->l;


fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;
```

```
/* Length of UDP */

ptr_courant =mallocEnreg();
ptr_courant->n = 318;
lire_n_hexa (1, tab);
ld = conv_hex (tab);
lire_n_hexa (1, tab);
ld = ld * 256 + conv_hex (tab);
sprintf (ptr_courant->v, "%d", len);
ptr_courant->l = strlen (ptr_courant->v);
lt = lt + 4 + ptr_courant->l;

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

/* Checksum */

ptr_courant =mallocEnreg();
ptr_courant->n = 319;
lire_n_hexa (2, ptr_courant->v);
ptr_courant->l= strlen (ptr_courant->v);
lt = lt + 4 + ptr_courant->l;

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

ld = ld - 8;

/* Data */

ptr_courant =mallocEnreg();
ptr_courant->n = 320;
if (ld != 0)
  {
    for (i = 1;i <= ld; i++)
      {

        lire_n_hexa (1, t);
        strcat(ptr_courant->v, t);
      }

    ptr_courant->l = strlen (ptr_courant->v);
    lt = lt + 4 + ptr_courant->l;
  }
else
{
  lt = lt + 4;
  ptr_courant->l = 0;
}
fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

return (lt);

}
/************************************************/
```

```
long trt_tcp (fin, ld)
struct enreg *fin;
int ld;
{
/* lit dans le fichier de trace, traite l'information lue, qui est le paquet
   tcp, remplit les champs du protocole utilise est tcp  */

int port;
int Hlen = 0;
int i;
char inter[500];
char aux[1000];
char t[1000];
char don_cod[1000];
struct enreg *mallocEnreg();

/* */

struct enreg *ptr_courant;
long lt = 0;

/* TCP */
/*******/
/*******/

/* SP */

ptr_courant =mallocEnreg();
ptr_courant->n = 216;
lire_n_hexa (1, inter);
port = conv_hex (inter);
lire_n_hexa (1, inter);
port = port * 256 + conv_hex (inter);
sprintf (t, "%d", port);
strcat (t, " ");
strcpy (ptr_courant->v, t);
ptr_courant->l = strlen (ptr_courant->v);
lt = lt + 4 + (ptr_courant->l);

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

/* DP */

ptr_courant =mallocEnreg();
ptr_courant->n = 217;
lire_n_hexa (1, inter);
port = conv_hex (inter);
lire_n_hexa (1, inter);
port = port * 256 + conv_hex (inter);
sprintf (t, "%d",port);
strcat (t, " ");
strcpy (ptr_courant->v, t);
ptr_courant->l = strlen (ptr_courant->v);
lt = lt + 4 + (ptr_courant->l);

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;
```

```
/* SeqN */

ptr_courant =mallocEnreg();
ptr_courant->n = 218;
lire_n_hexa (4, ptr_courant->v);
ptr_courant->l = strlen (ptr_courant->v);
lt = lt + 4 + (ptr_courant->l);

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

/* AckN */

ptr_courant =mallocEnreg();
ptr_courant->n = 219;
lire_n_hexa (4, inter);
ptr_courant->l = strlen (inter);
strcpy (ptr_courant->v, inter);
lt = lt + 4 + (ptr_courant->l);

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

/* Off = HL */

ptr_courant =mallocEnreg();
ptr_courant->n = 220;
lire_n_hexa (1, inter);
strcpy (aux, inter);
inter[1] = '\0';
strcat (inter, " ");
strcpy (ptr_courant->v, inter);
Hlen = (atoi(inter)) * 4;

ptr_courant->l = strlen (ptr_courant->v);
lt = lt + 4 + (ptr_courant->l);

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

/* Res */

ptr_courant =mallocEnreg();
ptr_courant->n = 221;
inter[0] = aux[1];
strcat (inter, " ");
strcpy (ptr_courant->v, inter);
ptr_courant->l = strlen (ptr_courant->v);
lt = lt + 4 + (ptr_courant->l);

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

/* Code */

ptr_courant =mallocEnreg();
```

```
ptr_courant->n = 222;
lire_n_hexa (1, inter);
strcpy (ptr_courant->v, inter);
ptr_courant->l = strlen (ptr_courant->v);
lt = lt + 4 + (ptr_courant->l);

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

/* Win */

ptr_courant = mallocEnreg();
ptr_courant->n = 223;
lire_n_hexa (1, inter);
port = conv_hex (inter);
lire_n_hexa (1, inter);
port = port * 256 + conv_hex (inter);
sprintf (t, "%d", port);
strcat (t, " ");
strcpy (ptr_courant->v, t);
ptr_courant->l = strlen (ptr_courant->v);
lt = lt + 4 + (ptr_courant->l);

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

/* Chk */

ptr_courant =mallocEnreg();
ptr_courant->n = 224;
lire_n_hexa (2, inter);
strcpy (ptr_courant->v, inter);
ptr_courant->l = strlen (ptr_courant->v);
lt = lt + 4 + (ptr_courant->l);

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

/* UP */

ptr_courant =mallocEnreg();
ptr_courant->n = 225;
lire_n_hexa (2, inter);
strcpy (ptr_courant->v, inter);
ptr_courant->l = strlen (ptr_courant->v);
lt = lt + 4 + (ptr_courant->l);

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;
                                /* traitement du champ option */
/* Options (s'il y en a) */
if (Hlen != 20)
  {
    ptr_courant =mallocEnreg();
    ptr_courant->n = 226;
    lire_n_hexa (Hlen - 20, inter);
    strcpy (ptr_courant->v, inter);
```

```
    ptr_courant->l = strlen (ptr_courant->v);
    lt = lt + 4 + (ptr_courant->l);

    fin->next = ptr_courant;
    ptr_courant->next = NULL;
    fin = ptr_courant;
  }
ld  = ld - Hlen;


/* Data */
/********/

ptr_courant =mallocEnreg();
ptr_courant->n = 227;
if (ld != 0)
  {
    for (i = 1;i <= ld; i++)
      {

        lire_n_hexa (1, t);
        strcat(ptr_courant->v, t);
      }

    ptr_courant->l = strlen (ptr_courant->v);
    lt = lt + 4 + ptr_courant->l;
  }
else
{
  lt = lt + 4;
  ptr_courant->l = 0;
}

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

return (lt);

}
/************************************************/

ecrire_buf (lt, debut)
int lt;
struct enreg *debut;

{
/* Lit les donnees de la liste chainee et les ecrit dans un buffer qui sera
   par la suite copie dans le fichier nadf par une fonction predefinie  */

struct enreg  *ptr_courant, *tmp;
int *ptr_lg;
unsigned short *ptr_id;
unsigned short *ptr_l;
char *ptr_rec_cour;

ptr_lg = (int *) ptr_rec;
ptr_rec_cour = ptr_rec + sizeof(int);

ptr_courant = debut;
do {
```

```
        if (ptr_courant->n == 320) {
                debut = debut;
        }

if ((ptr_courant->l) % 2 != 0) {
  ptr_courant->l = ptr_courant->l + 1;
  strcat (ptr_courant->v, " ");
  lt++;
}
ptr_id = (unsigned short *) ptr_rec_cour;
*ptr_id = ptr_courant->n;

ptr_rec_cour = ptr_rec_cour + sizeof(unsigned short);
ptr_l = (unsigned short *) ptr_rec_cour;
*ptr_l = ptr_courant->l;

ptr_rec_cour = ptr_rec_cour + sizeof(unsigned short);
bcopy (ptr_courant->v, ptr_rec_cour, ptr_courant->l);
ptr_rec_cour = ptr_rec_cour + ptr_courant->l;

tmp = ptr_courant;
ptr_courant = ptr_courant->next;
free(tmp);

} while (ptr_courant != NULL);
*ptr_lg = lt;
}
/*************************************************/

/*************************************************/

trt_paquet  (fd1, lec, tab)
char lec[];
char tab[];
int fd1;

{
/*lit d'abord les premiers champs du fichier de traces genere par etherfind,
  les traite puis les stocke dans une liste chainee, quand elle arrive au
  champ protocole, elle traite udp ou tcp selon le protocole trouve, enfin
  elle ecrit tous les champs se trouvant dans la chaine, dans le fichier
  NADF par l'intermediaire d'un buffer */

unsigned short flush = 1;
int rc;
long lt = 4;
long lg;
int ld;
int c;
int leng;
int i;
int Hlen;
int proto = 0;
char t[30];
char inter[25];
char pass[25];

/* */

extern char aux[];
```

```
struct enreg  *ptr_courant, *debut, *fin, *tmp;
struct enreg  *mallocEnreg();

/* remplissage du premier enregistrement et ensuite des enregistrements
   restants, l'ecriture dans le fichier NADF est a la fin de la fonction */

/* DA */

debut = mallocEnreg();
debut->n = 111;
strcpy (aux, "");
strcat (aux, lec);
strcat (aux, " ");
strcat (aux, tab);
strcat (aux, " ");
lire_n_hexa (4, inter);
strcat (aux, inter);
debut->l= strlen(aux);
lt = lt + 4 + debut->l;
strcpy (debut->v, aux);

debut->next = NULL;
fin = debut;


/* SA */

ptr_courant =mallocEnreg();
ptr_courant->n = 112;
lire_n_hexa (6, inter);
strcpy (ptr_courant->v, inter);
ptr_courant->l= strlen (ptr_courant->v);
lt = lt + 4 + ptr_courant->l;

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

/* FT */

ptr_courant =mallocEnreg();
ptr_courant->n = 113;
lire_n_hexa (2, inter);
strcpy (ptr_courant->v, inter);
ptr_courant->l= strlen (ptr_courant->v);
lt = lt + 4 + ptr_courant->l;

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

/* IP */
/******/
/******/
/* Ver */

ptr_courant =mallocEnreg();
ptr_courant->n = 114;
lire_n_hexa (1, inter);
strcpy (pass, inter);
inter[1] = '\0';
```

```
strcat (inter, " ");
strcpy (ptr_courant->v, inter);
ptr_courant->l = strlen(ptr_courant->v);
lt = lt + 4 + ptr_courant->l;

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

/* HL */

ptr_courant = mallocEnreg();
ptr_courant->n = 115;
inter[0] = pass[1];
strcat (inter, " ");
strcpy (ptr_courant->v, inter);
ptr_courant->l = strlen (ptr_courant->v);
lt = lt + 4 + ptr_courant->l;

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;


pass[0] = '0';
Hlen = conv_hex (pass) * 4;

/* ST */

ptr_courant =mallocEnreg();
ptr_courant->n = 116;
lire_n_hexa (1, inter);
strcpy (ptr_courant->v, inter);
ptr_courant->l = strlen (ptr_courant->v);
lt = lt + 4 + ptr_courant->l;

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

/* TL */

ptr_courant=mallocEnreg();
ptr_courant->n = 117;
lire_n_hexa (1, inter);
leng = conv_hex (inter);
strcpy (aux, inter);
lire_n_hexa (1, inter);
leng = leng * 256 + conv_hex (inter);
strcat (aux, inter);
strcpy (ptr_courant->v, aux);
ptr_courant->l = strlen (ptr_courant->v);
lt = lt + 4 + ptr_courant->l;

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

ld = leng - Hlen;

/* ID *** FLAGS * FRAGMENT OFFSET */
```

```
for (i = 118;i < 120;i++)
{
   ptr_courant=mallocEnreg();
   ptr_courant->n = i;
   lire_n_hexa (2, inter);
   strcpy (ptr_courant->v, inter);
   ptr_courant->l = strlen (ptr_courant->v);
   lt = lt + 4 + ptr_courant->l;


   fin->next = ptr_courant;
   ptr_courant->next = NULL;
   fin = ptr_courant;

}

/* Time *** Proto*/

for (i = 120;i < 122;i++)
{
   ptr_courant=mallocEnreg();
   ptr_courant->n = i;
   lire_n_hexa (1, inter);
   strcpy (ptr_courant->v, inter);
   ptr_courant->l = strlen (ptr_courant->v);
   lt = lt + 4 + ptr_courant->l;

   fin->next = ptr_courant;
   ptr_courant->next = NULL;
   fin = ptr_courant;
}

if ((inter[0] == '0') && (inter[1] == '6'))
        proto = 1;
else if ((inter[0] == '1') && (inter[1] == '1'))
     proto = 2;
     else proto = 3;

/* Hchk */

ptr_courant=mallocEnreg();
ptr_courant->n = 122;
lire_n_hexa (2, inter);
strcpy (ptr_courant->v, inter);
ptr_courant->l = strlen (ptr_courant->v);
lt = lt + 4 + ptr_courant->l;

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

/* IPS */

ptr_courant=mallocEnreg();
ptr_courant->n = 123;
conv_adr_ip (4, t);
strcpy (ptr_courant->v, t);
ptr_courant->l = strlen (ptr_courant->v);
lt = lt + 4 + ptr_courant->l;
```

```
fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

/* IPD */

ptr_courant=mallocEnreg();
ptr_courant->n = 124;
conv_adr_ip  (4, t);
strcpy (ptr_courant->v, t);
ptr_courant->l = strlen (ptr_courant->v);
lt = lt + 4 + ptr_courant->l;

fin->next = ptr_courant;
ptr_courant->next = NULL;
fin = ptr_courant;

/* IP OPTION (s'il y en a) */

if (Hlen != 20)
{
  ptr_courant=mallocEnreg();
  ptr_courant->n = 125;
  lire_n_hexa (Hlen - 20, inter);
  strcpy (ptr_courant->v, inter);
  ptr_courant->l = strlen (ptr_courant->v);
  lt = lt + 4 + ptr_courant->l;

  fin->next = ptr_courant;
  ptr_courant->next = NULL;
  fin = ptr_courant;
}

if (proto == 1)
        lg = trt_tcp  (fin, ld);
else if (proto == 2)
        lg = trt_udp (fin);
     else
{
  ptr_courant = debut;
  tmp = ptr_courant;
  ptr_courant = ptr_courant->next;
  free (tmp);
}


/* ecriture dans le fichier nadf */

lt = lt + lg;
ecrire_buf (lt, debut);
if ((rc = write_NADF (fd1, ptr_rec, flush)) != 0)
        printf("unable to write NADF record %d\n", rc);

}

/*************************************************/

copy_bytes(from, to, n)
char *from, *to;
int n;
```

```
{

/* copie les n bytes d'une zone "from" a une zone "to" */

   while (n > 0) {
        *to++ = *from++;
         n--;
    }
}
/*****************************************************/
```

Annexe F.

PROGRAMMES
DES
REQUETES RUSSEL

```
/************************************************/
/*                                              */
/*            La regle RUSSEL "DISPLAY"         */
/*                                              */
/************************************************/


global v:integer;

rule DISPLAY;
begin
        v := v + 1;
        println('Paquet numero: ', v);
                                /* incrementation de la variable globale v pour
                                   l'affichage du numero de paquet  */


        if (Proto %= '06' or  Proto %= '11') --> display_current fi;
                                /* Appel de la routine C "display_current qui
                                   affiche le contenu du paquet TCP ou UDP  */

        trigger off for_next DISPLAY
                                /* declenchement de la requete pour le paquet
                                   suivant  */
end;


init_action;
begin
        v := 0;
        trigger off for_next DISPLAY
end.
```

```
/************************************************/
/*                                              */
/*         La regle RUSSEL "RECONSTRUCT"        */
/*                                              */
/************************************************/


global i: integer;

/* */

rule RECONSTRUCT;
var j: integer;
    donnee_tcp, donnee_udp : string;

begin
               if (IPS %= '138.48.5.61' and IPD %= '138.48.5.58'
                                        and SP_tcp %= '23' )
               --> begin
                       if      present data -->
                               begin
                               convert(data, donnee_tcp);
                               /* Appel de la routine convert qui
                                  recoit une data hexadecimale et
                                  retourne une donnee lisible, si
                                  le paquet est du type TCP    */

                               print(donnee_tcp)
                               end;

                           present Data -->
                               begin
                               convert(Data, donnee_udp);
                               /* Appel de la routine convert qui
                                  recoit une Data hexadecimale et
                                  retourne une donnee lisible, si
                                  le paquet est du type UDP    */


                               print(donnee_udp)
                               end

                     fi;
                     i := i + 1;
                     if i mod 50 = 0 --> println('') fi
                               /* Passage a la ligne force tous les
                                  50 caracteres  */
                 end
       fi;
       trigger off for_next RECONSTRUCT
                               /* Declenchement de la requete
                                  RECONSTRUCT pour le prochain paquet*/


end;

/* */

init_action;
```

```
begin
        i := 0;
        trigger off for_next RECONSTRUCT
end.
```

```
/*************************************************/
/*                                               */
/* La table de correspondance des noms des champs */
/*                                               */
/*************************************************/

#define  tshort             10
#define  tint               11
#define  tlong              12
#define  tstr               13
#define  tunsignedint       14
#define  tunsignedshort     15
#define  tunkn     16


typedef struct   el_nadftab_str {
        char  ndata_name[16+1];         /* nom de l'audit data */
        int   ndata_typ;                /* type de l'audit data */


                } elem_tab_type;

#define max_nadf_data1  15 /* OK */


elem_tab_type nadf_tab1[max_nadf_data1] ={

         {"DA               " , tstr} /* 0 */
        ,{"SA               " , tstr} /* 1 */
        ,{"FT               " , tstr} /* 2 */
        ,{"Ver_ip           " , tstr} /* 3 */
        ,{"HL               " , tstr} /* 4 */
        ,{"ST               " , tstr} /* 5 */
        ,{"TL               " , tstr} /* 6 */
        ,{"ID               " , tstr} /* 7 */
        ,{"FFO              " , tstr} /* 8 */
        ,{"Time             " , tstr} /* 9 */
        ,{"Proto            " , tstr} /* 10 */
        ,{"Hchk             " , tstr} /* 11 */
        ,{"IPS              " , tstr} /* 12 */
        ,{"IPD              " , tstr} /* 13 */
        ,{"IpOption         " , tstr} /* 14 */
        };

#define max_nadf_data2  12

elem_tab_type nadf_tab2[max_nadf_data2] ={


         {"SP_tcp           " , tstr} /* 15 */
        ,{"DP_tcp           " , tstr} /* 16 */
        ,{"SeqN             " , tstr} /* 17 */
        ,{"AckN             " , tstr} /* 18 */
        ,{"Off              " , tstr} /* 19 */
        ,{"Res              " , tstr} /* 20 */
        ,{"Code             " , tstr} /* 21 */
        ,{"Win              " , tstr} /* 22 */
        ,{"Chk_tcp          " , tstr} /* 23 */
        ,{"UP               " , tstr} /* 24 */
        ,{"Tcp_Op           " , tstr} /* 25 */
```

```
        ,{"data             " , tstr} /* 26 */
        };

#define max_nadf_data3  5

elem_tab_type nadf_tab3[max_nadf_data3] ={


         {"SP               " , tstr} /* 27 */
        ,{"DP               " , tstr} /* 28 */
        ,{"Length           " , tstr} /* 29 */
        ,{"Chk_udp          " , tstr} /* 30 */
        ,{"Data             " , tstr} /* 31 */
        };
```

```
/*************************************************/
/*                                               */
/*    Fichier de description des audits data     */
/*                                               */
/*************************************************/


A 1.0
B UNIX_System_V lhasa 4.1ES 2 3B2
D 1.0

1 111
2 tstr
3 tstr
4 DA
5 destination_address
1 112
2 tstr
3 tstr
4 SA
5 source_address
1 113
2 tstr
3 tstr
4 FT
5 ft
1 114
2 tstr
3 tstr
4 Ver_ip
5 version
1 115
2 tstr
3 tstr
4 HL
5 header_length
1 116
2 tstr
3 tstr
4 ST
5 service_type
1 117
2 tstr
3 tstr
4 TL
5 total_length
1 118
2 tstr
3 tstr
4 ID
5 identification
1 119
2 tstr
3 tstr
4 FFO
5 flags_and_fragment_offset
1 120
2 tstr
3 tstr
4 Time
```

```
5 time_to_live
1 121
2 tstr
3 tstr
4 Proto
5 protocol
1 122
2 tstr
3 tstr
4 Hchk
5 header_checksum
1 123
2 tstr
3 tstr
4 IPS
5 source_ip_address
1 124
2 tstr
3 tstr
4 IPD
5 destination_ip_address
1 125
2 tstr
3 tstr
4 IpOption
5 ip_options_and_padding
1 216
2 tstr
3 tstr
4 SP_tcp
5 source_port
1 217
2 tstr
3 tstr
4 DP_tcp
5 destination_port
1 218
2 tstr
3 tstr
4 SeqN
5 sequence_number
1 219
2 tstr
3 tstr
4 AckN
5 acknowledgement_number
1 220
2 tstr
3 tstr
4 Off
5 offset
1 221
2 tstr
3 tstr
4 Res
5 reserved
1 222
2 tstr
3 tstr
4 Code
5 code,_offset_and_reserved_=_HL
```

```
1 223
2 tstr
3 tstr
4 Win
5 window
1 224
2 tstr
3 tstr
4 Chk_tcp
5 checksum
1 225
2 tstr
3 tstr
4 UP
5 urgent_pointer
1 226
2 tstr
3 tstr
4 Tcp_Op
5 Option_tcp
1 227
2 tstr
3 tstr
4 data
5 data_of_tcp_packet
1 316
2 tstr
3 tstr
4 SP
5 source_port_udp
1 317
2 tstr
3 tstr
4 DP
5 destination_port
1 318
2 tstr
3 tstr
4 Length
5 length
1 319
2 tstr
3 tstr
4 Chk_udp
5 checksum
1 320
2 tin
3 tstr
4 Data
5 data_of_udp_packet
```

```
/*************************************************/
/*                                               */
/*         La routine C "display_current"        */
/*                                               */
/*************************************************/


#include <stdio.h>
#include "/home/patate/users/folon/amo/asax/portable/predef.h

#include "nadf.tab.h"
                /* Table de correspondance des identifiants ainsi que leurs
                   types  */

#define TAILLE 2000

/* */

void *display_current(m)
char *m;
{
    display_current1(Curr_Rec_Addr);
    printf("\n");
}

/* */

display_current1(rec_addr)
char *rec_addr;
{
    int nb;
    short len;
    short id;
    int  type;
    int i;
    int j;
    int k;
    unsigned char *pa, c;
    char  buf[TAILLE];
    char  *rec_cur_ptr;
    int   *rec_len_ptr;
    short *shp, *rec_sht_ptr;
    long int *rec_longInt_ptr;
    char *malloc(), *ctime(), *pstr, *pstr1;

/* */

  rec_len_ptr = (int *) rec_addr;
  rec_cur_ptr = rec_addr + 4;

                /* Convertion des pointeurs et leurs positionnements au debut
                   de l'enregistrement data du paquet dans le fichier NADF */


  nb = *rec_len_ptr - 4;
  while (nb >0)
              {
                shp = rec_sht_ptr = (short *) rec_cur_ptr;
                id = *rec_sht_ptr;
```

```
printf("\n");

/* Test des identifiants  */

if ( (id >= 111) && (id <= 125)) {
        printf("%s[", nadf_tab1[id - 111].ndata_name);
        type = nadf_tab1[id - 111].ndata_typ;
        }
else  if ( (id >= 216) && (id <= 227)) {
        printf("%s[", nadf_tab2[id - 216].ndata_name);
        type = nadf_tab2[id - 216].ndata_typ;
        }
      else  if ( (id >= 316) && (id <= 320)) {
                if (id >= 319)
                        id = id;
                printf("%s[", nadf_tab3[id - 316].ndata_name);
                type = nadf_tab3[id - 316].ndata_typ;
                }

printf("%-4d, ", id);

/* Avancement des pointeurs et lecture de la longueur du champ
   valeur pour un identifiant  */

rec_sht_ptr++;
len = *rec_sht_ptr;

/* Affichage de l'identifiant, de la longueur et de la valeur
   selon un format precis  */

printf("%-2d] = ", len);
nb = nb-4-len;
rec_cur_ptr = rec_cur_ptr+4;
switch(type) {

  case tshort :
  case tint   :
  case tlong  :
  case tunsignedint :
  case tunsignedshort :

    switch(len) {
        case 1:
        case 2:  rec_sht_ptr = (short *) rec_cur_ptr;
                 printf("%d",*rec_sht_ptr);
                 break;
        case 4:  rec_len_ptr = (int *) rec_cur_ptr;
                 if (id == 2 || id == 5 || id == 122) {
                        pstr = malloc(24);
                        pstr1 = ctime((long *) rec_cur_ptr);
                        copy_bytes(pstr1,pstr,24);
                        printf("%s", pstr);
                        free(pstr);
                 } else {
                        printf("%d",*rec_len_ptr);
                 }
                 break;
        case 8:  rec_longInt_ptr = (long int *) rec_cur_ptr;
                printf("%ld",*rec_len_ptr);
                 break;
       default:
```

```
                printf("integer with len <> 2, 4, 8 " );
                break;
                }
        break;

            case tstr    :

         if (id == 320 || id == 227) {
            int len11;

            /* Appel de la sous-routine C "sub_convert" qui nous
               retourne la donnee convertie du buffer et sa longueur
               len11  */

            len11 = sub_conv(rec_cur_ptr, buf, len);
            buf[len11] = '\0';

            /* Affichage de la valeur de l'enregistrement  */

            printf("%s", buf);
         }
        else {
            copy_bytes(rec_cur_ptr,buf,len);
            buf[len] = '\0';
            printf("%s", buf);
         }
         break;

            default:

         pa = (unsigned char *) rec_cur_ptr;
         i = 1;
         while ( i <= len )
{
          j = 1;
          while  ( (j <= 5) && (i <= len) )
          {
             k = 1;
             while ( (k <=4) && (i <=len) )
              {
               c = (unsigned char) *pa;
               if (c < 10)
                   printf("0%x",*pa);
               else
                   printf("%x", *pa);
               pa++;
               i++;
               k++;
               }
             j++;
          }
          printf("");

         }
         break;


                } /* fin switch type */

        /* avancement du pointeur courant sur l'enregistrement suivant*/
```

```
                rec_cur_ptr = rec_cur_ptr+len;
                printf("  ");
                }
}
```

```
/*************************************************/
/*                                               */
/*               La routine C "convert"          */
/*                                               */
/*************************************************/


#include "/home/patate/users/folon/amo/asax/portable/predef.h"

/*************************************************/
int conv_hex (hex)
char hex[];
{
/* Convertion, recoit un hexadecimal dans hex et retourne un entier */

int val;
/*
int i = 0;

if hex[i] = ' '
        i++;
*/
if ((hex[0] <= '9') && (hex[0] >= '0'))
   val = hex[0] - '0';

else if ((hex[0] <='f') && (hex[0] >= 'a'))
        val = hex[0] - 'a' + 10;

     else if ((hex[0] <='F') && (hex[0] >= 'A'))
                val = hex[0] - 'A' + 10;

          else {
                printf("non hexa string %s\n", hex);
                exit(1);
}


if ((hex[1] <= '9') && (hex[1] >= '0'))
   val = val * 16 + hex[1] - '0';

else if ((hex[1] <= 'f') && (hex[1] >= 'a'))
        val = val * 16 + hex[1] - 'a' + 10;

     else if ((hex[1] <='F') && (hex[1] >= 'A'))
                val = val * 16 + hex[1] - 'A' + 10;

          else {
                printf("non hexa string %s\n", hex);
                exit(1);
                }

return (val);
}
/*************************************************/

sub_conv (s, t, lg)
char *s;
char *t;
int lg;
```

```
{

/* Convertir une donnee hexa pointe par s de longueur lg et la stocker dans
   la zone t et retourne sa longueur lg1  */

  int i, j, nas, lg1;
  char tab[5], *t1;

/* */


t1 = t;
lg1 = 0;
while ( lg >= 3) {
        for (j = 0;j < 3;j++) {
          tab[j] = *s++;
          lg--;
          }

        tab[j] = '\0';
        nas = conv_hex(tab);
        if (isprint(nas))
                *t1 = ((char)nas);
        else {
             *t1++ = tab[0];
             *t1 =   tab[1];
        }
        t1++;
        lg1++;

      }

return (lg1);
}
/*****************************************************/
/*****************************************************/

convert (m)
char *m;
{

/* Positionnement dans la zone memoire creee pour la lecture des arguments de
   la routine C  */


hole **h;
unsigned short *sh;
int lg, lg1;
char *s, *t;

/* */

h = (hole **) m;
lg = lenstr (*h);

if (lg != 0) {

        sh = (unsigned short *) malloc (lg);

        s = (char *) *h;
```

```
        s = s + sizeof(short);

        m = m + sizeof(int);
        h = (hole **) m;

        **h = (hole) sh;
        t = (char *) sh;
        t = t + sizeof(short);

        lg1 = sub_conv (s, t, lg);
        *sh = lg1;

        }
else {
        sh = (unsigned short *) malloc (2);
        m = m + sizeof(int);
        h = (hole **) m;
        **h = (hole) sh;
        *sh = 0;
        }

}

/*************************************************/
```